MASTER IN COMPUTER SCIENCE

Reasoning on Gene Regulatory Networks using Constraint Logic Programming

Herbin, Geoffroy

Award date:
2018

Awarding institution:
University of Namur

Université de Namur
Faculty of Computer Science
Academic Year 2017–2018

Reasoning on Gene Regulatory Networks using Constraint Logic Programming

Geoffroy Herbin

Supervisor: _____ (Signed for Release Approval - Study Rules art. 40)
Jean-Marie Jacquet

A thesis submitted in the partial fulfillment of the requirements
for the degree of Master of Computer Science at the Université of Namur

Acknowledgments

I would like to thank my supervisor, Prof. Jean-Marie Jacquet, who first introduced me to the constraint logic programming, and offered me to work on this thesis' topic. I am extremely grateful for his ideas, advices, observations and guidelines that helped me throughout this thesis.

I am very grateful to my family for their continuous unconditional amazing support, care and encouragements. They have been a precious help not only on this thesis, but also during the past two years.

I express as well my sincere gratitude to all of my friends who supported me since this adventure has started.

Finally, I couldn't be more thankful to Ana for her extreme patience, her tolerance, her solicitude, and her encouragements.

Abstract

Gene Regulatory Networks (GRN) inference is of great interest for biologists, considering the substantial information these networks can provide. This thesis shows how the GRN inference can be translated into a Constraint Satisfaction Problem (CSP), and benefit from the Constraint Logic Programming (CLP) paradigm.

Starting from current known modeling techniques, this thesis details how to model a GRN inference problem as a CSP. Based on this theoretical result, the prototype of a tool capable of reasoning over GRN is built as a Web application, back-end and front-end.

This tool aims at allowing the biologists to infer GRN from experimental data, but also assess hypotheses on parameters of the networks. Required degrees of freedom, based on the biological modeling and assumptions, are provided to the user.

Different implementations of the core of the CSP, part of the back-end, are provided, and their performances are assessed thanks to a systematic tests framework developed. This assessment helps defining heuristic allowing to automatically or manually choose, in the tool, what methodology using depending on the user inputs or expectations. As an illustration, a case-based application of the tool is provided, the *simplified* lac operon network.

Keywords

Gene Regulatory Networks, Constraint Logic Programming, Prolog

Contents

Introduction

Personalized medicine – a type of medical care in which treatment is customized for an individual patient – is probably one the most promising fields of medical research. To that end, massive amount of work is still needed, sharing a common interest: better understand the human genome, constituted of genes, these basic physical and functional biochemical units.

Understanding human genome passes by understanding genome in general – the complete set of genes or genetic material present in a cell or organism –, and how it affects the development of organisms, the response to external stimuli,...

In that context, it is fundamental to deeply understand how the genes and their products, the proteins, interact on each others in an environment, and to learn the mechanisms and influences governing the way proteins are produced. This is the very base of the gene regulatory networks (GRN), which attempt to concentrate all these interactions and mechanisms called gene expression. This information is later used, for instance, in the context of personalized medicine.

There are multiple ways of building those networks, according to different modeling techniques. Some techniques are quantitative, aiming at providing to biologists the exact values of expression levels or gene products concentration levels, while others, qualitative, help defining the overall evolutions of these concentration and expression levels. These qualitative techniques allow in particular to compensate the lack of experimental or noisy data. One of the most known and used models is the Boolean network, a logic modeling technique described by Kauffman in [36] or Thomas in [62, 63, 64].

While these techniques are used in practice to reason on the GRN and the dynamic of the genes interactions, it remains a difficult task to easily assess new biological hypotheses, confirm modeling choices, and compare different possibilities of GRN that all could explain experimental results. Practically, there is quite a lack of tools offered to the biologists to properly and efficiently reason on the gene regulatory networks.

This thesis aims at suggesting a prototype of such a tool, usable by biologists, and based on the powerful paradigm of constraint logic programming.

The constraint logic programming is particularly well suited to solve efficiently combinatorial search problems. The GRN inference can be considered as such a problem, with many unknowns – constrained by biological modeling-related assumptions –, and uncertainty linked to biological hypotheses and experimental data.

The first chapter of this thesis focuses on giving the biological notions required to understand the gene regulatory networks. Graphical examples of GRN are provided, and their utility is discussed in more details, in order to fully perceive the interest of the different modeling techniques. Some of these techniques are then elicited, with first a presentation of the required mathematical framework generally accepted and used in the modeling, then the presentation of the data-driven methods, logical methods, and differential equations methods. The interests of these various methods are discussed. A concrete example of a real gene regulatory network is shown in practice, the lac operon. This enlightens the utility GRN have, and how they can be used in practice. It confirms also that GRN are not per se the final output, but rather a

tool to help in understanding biochemical reactions and behaviors.

The second chapter introduces largely the constraint logic programming paradigm used along this thesis. It describes first the idea behind the constraint programming paradigm, then gets into the details of constraint satisfaction problem (CSP) solving: the constraint propagation, the search, the modeling and the labeling. As constraint programming relies on similar notions as logic programming, the opportunity to *merge* the two is presented, ending up with the constraint logic programming (CLP), and several tools currently used. A reminder of the logic programming – without constraints – is shown, then a more detailed CSP resolution using CLP is graphically detailed. Finally, the nominal steps of a CLP resolution are described: variables elicitation, constraints elicitation and labeling.

The third chapter presents how the gene regulatory network inferring can be seen as a constraint satisfaction problem. After a theoretical background presentation, going deeper in the biological-related modeling technique followed – based on Boolean networks –, the chapter is structured as the CSP resolution: first the variables modeling, then the constraints modeling, and finally the labeling techniques are detailed in the specific context of GRN. A last section also discusses the current work related to constraints and GRN models inference in the literature.

The fourth chapter focuses on the suggested tool prototype, and details the main choices in terms of general architecture – a Web application – and technologies. Based on that choices and on the modeling of the third chapter, a domain specific language is introduced. This serves to describe the network to reason on, and the expected services awaited by the user from the tool. This is largely covered in the last section of the chapter.

The fifth chapter goes into the details of the development of the back-end in Prolog, the programming language extensively used in this thesis. A presentation of the back-end's layered architecture is given, followed by the implementation of its core: the constraint logic program. Different modeling and labeling methods are implemented and assessed. This comparison, using a developed systematic tests framework, aims at opening avenues for heuristic developments and expert systems. Other features developed in the scope of the modeling presented in Chapter 3 (p33), are also presented and tested from a user perspective, setting basic use cases and assessing the results from the back-end computation. The sixth chapter succinctly presents the front-end developed, extensively used during the development. In particular, it is shown how to input data, request an inference and visualize the results, specifically when several solutions are provided.

The seventh chapter then shows a case-based application of the tool. Reusing the lac operon example introduced at the very beginning, two use cases are shown: the first one considering very few experimental data while the network is almost fully described, and the second one with the complete set of experimental data, but no interactions between genes known.

Finally, the eighth chapter aims at summarizing perspectives for future work that are raised in the thesis. Different aspects are covered, from the modeling and CLP themselves to the tool prototype development and uses.

Numerous code extracts are provided along the thesis, or referred to in Appendix, as material support. The complete source code, log files and utility scripts are available upon request to the author.

Chapter 1

Context

1.1 Introduction

This chapter introduces the reader with the real-world application context of this thesis, the gene regulatory networks. Firstly, explanations of several key notions are given, from the gene itself and its products up to the gene regulatory networks. These notions are explained, pictures when required, and real-life example are cited. Secondly, the interest of these gene regulatory networks and the way they are built, or rather modeled, is discussed. Several modeling techniques are presented in more details as they are – sometimes extensively, as for the logical modeling methods – used later in this thesis. Thirdly, a concrete and well-known example of real gene regulatory network, the *lac operon*, is introduced and sums up the chapter. This example allows showing how a gene regulatory network (re)acts in its environment, and how it can be described.

1.2 Biological notions

1.2.1 Gene and protein

A gene is considered to be a basic physical and functional biochemical unit, as expressed in [79]. The genes are made up by the DNA, a double helix formed by base pairs attached to a sugar-phosphate backbone. A graphical representation of the DNA is given in Figure 1.1 (p4). The bases are adenine (A), guanine (G), cytosine (C) and thymine (T). Those four bases help storing the information about any organism, the same way the alphabet helps defining the words of a language. DNA stands for deoxyribonucleic acid.

The genes act as a recipe to produce molecules of a specific type: the proteins. Among the proteins produced are the amino acids, and the genes that code the synthesis of the amino acids are known as "structural genes". The proteins are large, complex molecules that play critical roles in organisms. Specifically for the human body, the proteins do most of the work in the cells, and are required for the structure, function, and regulation of the body's tissues and organs. More information about the DNA can be found in [81].

1.2.2 Gene expression

Gene expression is the process by which the genetic code of a specific gene is used to direct protein synthesis, and produces the structures of the cell. Production of proteins from the gene code follows a complicated path consisting of two major steps: the transcription and the translation. Together, the transcription and translation are known as gene expression. The interested reader will find much more information on this process in online sources such as [19].

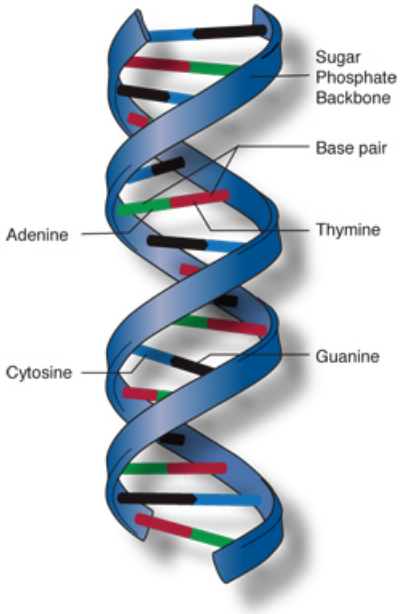

FIGURE 1.1: DNA representation

1. **Transcription** : the information stored (or coded) in a gene DNA is transfered to another similar molecule called RNA (ribonucleic acid) by the enzyme RNA polymerase. RNA have slightly different chemical properties. The type of RNA that contains information for making of a protein is called messenger RNA, usually referred to as mRNA. Two other kinds of RNA are also produced, the transfer RNA and ribosomal RNA, which both participate in the translation process, as shown in Figure 1.2 (p4).

2. **Translation** : it produces the protein from the previously built RNA.

The details about how transcription and translation actually work have limited interests in the context of this work, provided that the basic principle of gene expression is understood. More details about this process are given in numerous resources, such as in [18].

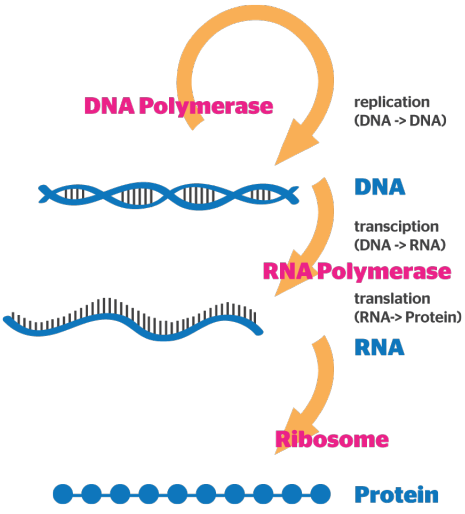

FIGURE 1.2: Gene Expression

Considering a structural gene, a gene that codes an amino acid, two major kinds of components can be observed :

- Exons : they code for amino acids, and collectively determine the amino acid sequence of the protein product. The exons are in fine represented in mature mRNA molecules.

- Introns : they are the portions of the gene that don't code for amino acids. They are removed from the mRNA molecule before translation.

1.2.2.1 Gene Control regions

Several specific regions of a gene are remarkable, as they play a key role in this gene expression.

1. **Start site**: the localization of the start for the transcription process.

2. **Promoter**: a region upstream of the gene. It is not transcribed into mRNA, but will rather control the gene transcription, allowing the binding of transcriptional factors.

3. **Enhancers**: regions where some transcription factors called activators will bind in order to increase the rate of transcription. Some enhancers are conditional: they can only work in the presence of other factors.

4. **Silencers**: regions where some transcription factors called repressors will bind in order to decrease the rate of transcription.

1.2.3 Gene regulation

As indicated in [18], gene regulation is the name given to the processes that control how genes expression happens.

In multicellular organisms, cells in different tissues, organs,... differentiate and become specialized by making different sets of proteins, whereas all the cells in the organism have the same genome (with a couple of exceptions). Furthermore, the amount of genes in different organisms is relatively constant across different scales of organisms complexity : the baker's yeast, *Saccharomyces cerevisiae*, so tiny, has indeed almost 6000 genes, which is about a quarter of the amount of genes that constitute a sophisticated human genome. As described in [29], it can be stated that the diversity and complexity of life does not arise from a disparity in the number of available basic components, but rather from the nature and dynamics of the interactions between those available components.

The gene expression can be regulated by various processes whereas the goal of the gene regulation is to control the amount and nature of the genes products, mainly proteins. This control is handled at numerous levels, thanks to other regulatory proteins, as indicated in [49].

As stated in [37], four main sources or levels of regulation of the gene expression can occur:

- **DNA rearrangement** which is rather funky and not really frequent. This can happen for the immune system, for instance.

- **Transcription** which is the most frequent. The regulation of the gene expression can happen when the transcription of the RNA is initiated.

- **Translation**: when mRNA is converted to proteins. That is not automatic, and is important.

- **Post-translational**: the proteins can be inactive in some specific conditions. Even if they are created, the proteins do not express themselves.

At some point, all of those four regulation levels happen, but the most crucial is at the level of transcription, when the DNA in transcripted to RNA (see Figure 1.2 (p4)). These proteins really bind to the DNA and send signals that will further (hence indirectly) control the rate of the gene expression.

The concept of gene regulation is related to the *operon*: an operon is a collection of genes that are regulated together. Genes in an operon are transcribed as a group, and have a single promoter. Operons are quite common in bacteria or prokaryotes, although quite rare in eukaryotes such as humans. Each operon contains regulatory DNA sequences, which act as binding sites for regulatory proteins that will *promote* (or *activate*) or *inhibit* transcription, as described in [28]. Usually, operons control important biochemical processes.

1.2.4 Gene Regulatory Networks - GRN

Measuring directly the interactions between components is quite complicated within live cells. However, measuring the components presence in a defined volume is much easier, and recent technological improvements have decreased the cost of such measurements ([20, 38, 57], or discussion in [10]), providing the motivation to reconstruct computationally those interactions and their structures.

Gene Regulatory Network is usually described as the set of all regulatory transcription interactions in a cell. Typically, a GRN is represented as a graph with edges connecting regulators to regulated genes, the nodes of the graph. Several examples are given in Figures 1.3 (p7), 1.4 (p7) and 1.5 (p7), for different networks and as a schematic view.

A definition of a gene regulatory network is given in [10]: *We call a network that has been inferred from gene expression data a "gene regulatory network," briefly denoted as GRN.*

A notion often related to the GRN is the data *inference*, setting the data retrieved from experience in the center of the analysis. In the following section, it will be shown that several modeling techniques, although based on experimental data, go way beyond simple data inference in order to cope with inherent experimental noise.

Usually, the transcriptor factor (TF, regulator) can be known, while the target genes (TGs, regulated genes) are mostly unknown. Another idea is simply to consider each gene as a potential TF and TG, and consider that all the genes in a system may be regulators and regulated.

The main focus of this work is on the regulation during the transcription, as it is the most frequent source of gene regulation (see Section 1.2.3 (p5)). However, other mechanisms happen, providing other means to regulate the gene expression. Those mechanisms are currently biologically less well explored, while it is believed that their effects may be as prevalent as the transcriptional control on the genes expressions. That is, while a gene may have no effect on the expression of another at RNA transcription level, it could be important for the protein expression. These aspects are out of the scope of this work, see Chapter 8 (p105).

1.3 Modeling Gene Regulatory Networks

Proteins are the final product of the process of gene expression ([29]); and the control of the gene expression is applied through the action of the gene products themselves.

The improvement in the field of gene expression measurements makes it possible to simultaneously measure the level of expression of several genes. Using *reverse engineering* techniques, one can reconstruct the interactions between genes (including their products) based on the measured data. This is one kind of possible GRN modeling. Many methods were developed in order to model the gene regulatory networks, that will be briefly described in this section, based on [14] and [29].

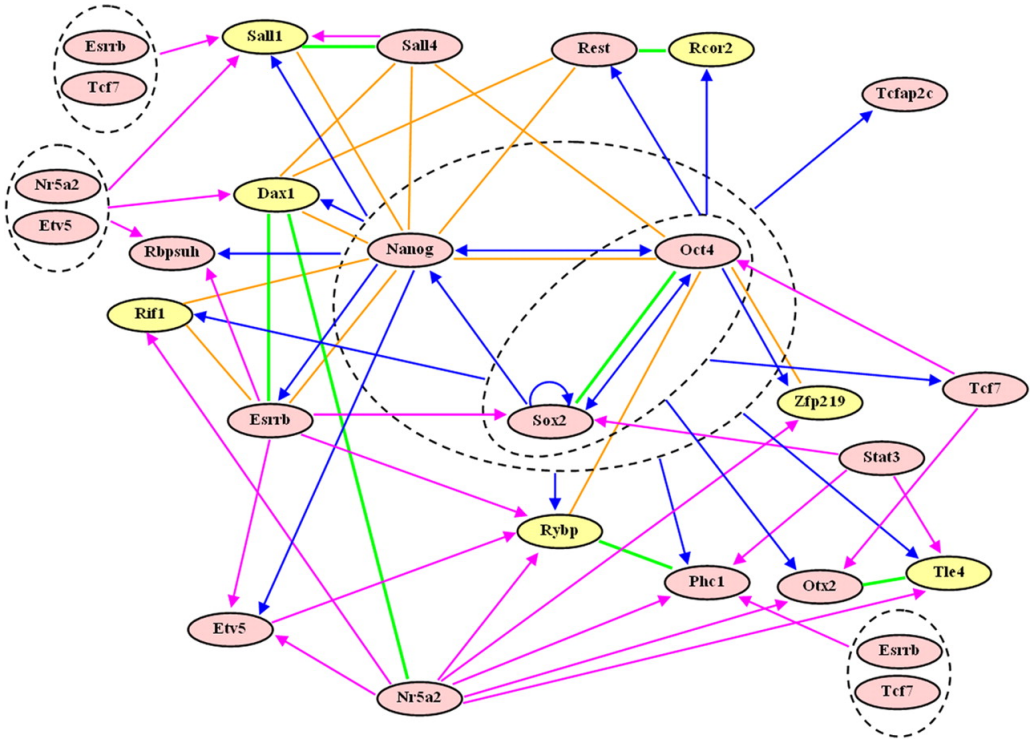

FIGURE 1.3: From [43], Gene Regulatory Network of Transcription Factor family NFKB[1] in human

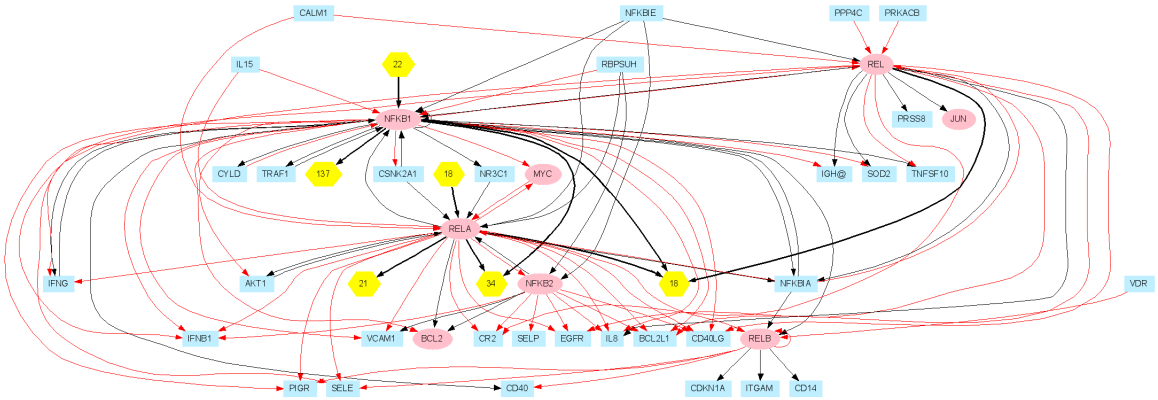

FIGURE 1.4: From [86], Gene Regulatory Network in mouse embryonic stem cells

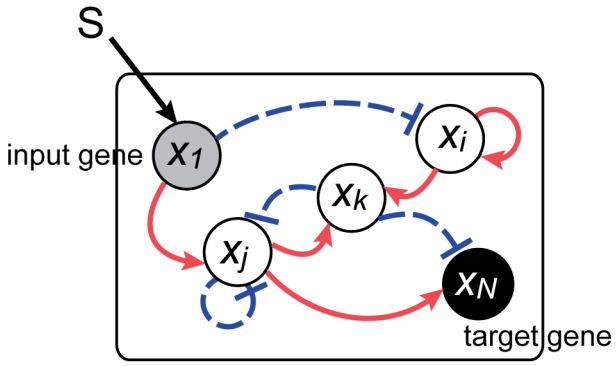

FIGURE 1.5: From [31], Schematic view of a gene regulatory network

A first important consideration is that, as elicited in [11], there is most likely not one method better than all the others, but rather a number of methods that result in an overlapping spectrum having the potential to infer similar biological information. That is, to get the global overview of a network, several viewpoints are most likely needed.

The interested reader can consult other resources in the literature, starting with [29] and [73], for more in-depth information.

1.3.1 Utility

Depending on the availability of empirical data (related to gene expression) and the desired level of abstraction, different levels of modeling can be used, as shown in the Figure 1.6 (p8) from [14]. It represents a hypothetical gene network. On the left are multiple levels of gene regulations, while the abstraction on the right uses the information to specify interactions between genes only. The nature of the interactions is either inducing (arrow) or repressing (dull end).

The different levels of abstraction represented on the left on Figure 1.6 (p8) serve typically different purposes, from simple hypothesis testing to extensive quantitative work.

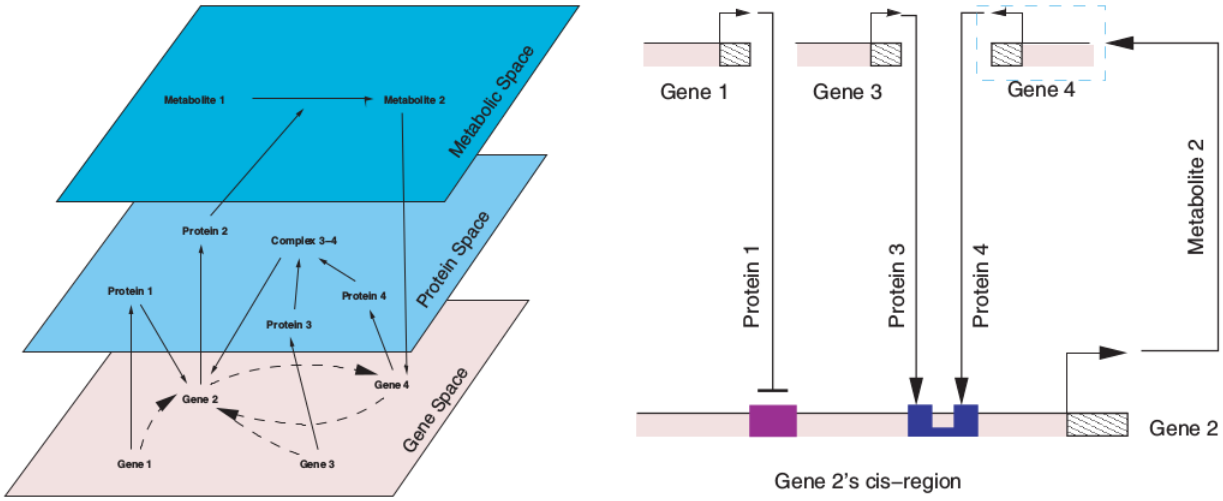

FIGURE 1.6: Different GRN modeling level

Per se, the inference of a GRN is not the ultimate result, rather a help in solving different biological and biomedical problems, which could be split, as defined in [10], in the different following categories.

1.3.1.1 Causal map of interactions

The gene regulatory networks can be understood as *maps* of the interactions between the genes (or proteins, or other gene products). Considering this map, the GRN can be used to derive new biological hypothesis about those interactions, specifically during the transcription of the mRNA. The GRN corresponds (or are supposed to correspond) to real binding events between genes, and it is therefore possible to reason on those models. This remains however a challenging tasks as the models usually do not take into account all possible interactions with other intermediate genes – including unknown gene products. Several promising results are compiled in [10].

1.3.1.2 Experimental design and perturbation experiments

The gene regulatory networks can precede lab experiments and guide the experimental design. More specifically, many experiments have as main objectives to deliver observational data: the experiments "observe" the system as it is, without any intervention or perturbation. Knowing

the part of the GRN to be observed allows to design experiments purposefully, where dedicated perturbations could be conducted to favor the system response, hence improving the efficiency of the experiments.

1.3.1.3 Networks as biomarkers

GRN, or part of them, can also be used in order to build markers based on several genes instead of individuals. As stated in [10], this is possible as the hallmarks of some complex diseases, such as cancer, are represented by pathways in which genes are actively interacting. Network-based biomarkers can help for those types of disorders, where there is a need to consider interaction changes instead of individual gene changes, which neglects the interactions between genes.

1.3.1.4 Comparative network analysis

Building GRN's and performing statistical comparison between them allows to learn about the interactions changes across different conditions (disease, physiological changes,...), hence improving the knowledge of biomedical processes. A subsequent need to achieve this comparative analysis is to establish free-access databases of the gene regulatory networks.

1.3.1.5 Network medicine and drug design

A better knowledge about the interactions between genes/gene products allows a better targeting during drug development. The article in [10] mentions for instance different studies eliciting concrete examples of the use of GRN's in drug development.

1.3.2 Empirical properties

Many researchers work on deriving networks from experimental gene expression data. The assumptions already validated by prior work can be used to ease the modeling tasks, and constitute modeling rules to follow. In particular, three biological properties are considered:

- **Topology**: it defines the connections between nodes, and it can be a starting point for modeling. It has been observed that the topology of GRN is usually quite *sparse*, meaning that there is solely a small number of edges per node. This number of edges is much smaller than the total number of nodes, and the graph resulting (network) is therefore far from complete. This property can be used to prune the search space during inference.

  Also, the frequency distribution of the connectivity of nodes seems to be better described by $P(k) = k^{-g}$ where k is the degree of the node and g is some specific constant. Such a network is called scale-free, and its connectivity gives two interesting properties: *the availability to reach rapidly any node*, from any starting node, and *the emergence of hubs*(nodes connected to many others), which are central in the network - involved in many regulation interactions. More information on such scale-free graph are available in [54]. These hubs correspond to highly central nodes in the gene network, i.e. genes that do a large amount of the overall regulation.

- **Robustness**: Real gene networks are very robust to fluctuations in their parameter values, which can be achieved only by specific topology choices. For instance, this fluctuation includes quite an insensitivity to variations in molecular concentrations at a certain period of time (organism development). The scale-freeness property is also a key to ensure the robustness of the gene network.

- **Noise**: As many natural phenomena, biochemical reactions are stochastic[2] per nature. Several modeling methods handle this stochasticity introducing discretization, or use real stochastic techniques, see Section 1.3.5 (p11). Others do not really handle this behavior.

1.3.3 Mathematical framework

Most of the GRN modeling techniques, based on *Reverse Engineering* principle, rely on mathematical material that is introduced in this section.

The following definition is taken from [29]: a directed network, or a graph, is a pair (V, E), where V is a finite set of nodes, and E is a set of edges, or arcs, connecting those nodes. If I is a set indexing the nodes, the set of edges is a subset of the Cartesian Product so that $E \subseteq I \times I$, with element (ij) indicating the presence of an edge between node i and node j. An *undirected* network is a network where the edge set is symmetric under swapping the indices of the nodes.

This definition of the network will be used in subsequent modeling techniques.

In the gene regulation domain, the nodes represent any compound of the system that interacts – or not – with others. It is therefore quite broad. In the following chapters, the mention to "gene" will refer either to a gene, a gene product (such as a protein), or any other entity that is part of the network.

In the models presented, the values linked to the nodes, as defined here above, denote the genes expression level – or as per previous remark, compounds expression level or compounds product concentration level (see Section 3.4.1 (p36)). The edges, however, are given quite different semantics according to the model techniques applied. In this context, the usual technique is to add a weight on each edge. This weight can have different meaning in the literature: a probabilistic existence factor, as used in Bayesian networks (see [73]), a threshold of edge activation, such as in [50], the existence confidence of the link, as expressed in [15], the impact of the source gene on the concentration level of the sink gene, etc.

1.3.4 Data-Driven methods

A first class of method to infer GRN uses a fully connected weighted network, and associate to the weight of the edge (ij) the estimated dependence of the gene j to the gene i.

The implementation is relatively simple, providing the data are accurate enough. Some techniques exist in order to normalize the experimental data, and take into account the inevitable noise of such data. Although it is of major interest for all subsequent modeling techniques, the normalization of the data is out of the scope of this work.

1.3.4.1 Regression-based methods

One of the most popular approaches in the context of Data-Driven methods is to predict a variable based on the others, using *simple* linear regression. The slope indicates therefore the strength of the dependence. Applied to the gene regulation network, the network weights are computed by regressing each gene in turn against all others, see [29]. That is, for every gene g, the expression level in sample i of g, x_{gi} is found by solving the regression equation

$$x_{gi} = \sum_{j \neq g} w_j x_{ji} + \epsilon_i$$

where ϵ_i integrates the noise on experimental data. The weight w_j is then associated with the edge (jg), between gene j and gene g. An important consequence of using the regression technique is the intrinsic directivity given to the network.

[2] Denotes something having a random probability distribution or pattern that may be analyzed statistically but may not be predicted precisely.

Two examples of tools using this techniques are TIGRESS software, which description is found in [25], and GENIE3, which is found in [72].

This method also allows dealing with time series data, hence providing predictive capability, if used such as the gene expression of g at time t is regressed against the expression data of the others genes at time $t-1$.

The equation introduced above can incorporate more specific terms than the ones indicated to reflect well-known biological reactions, such as the influence of the kainate or other bias, see [73].

Generally, the regression-based methods induce more intense computation, but provide major advantages on other data-driven methods such as the predictability – the ability to predict the expression level of remaining genes or for later time[3], based on the experimental data.

1.3.4.2 Other data-driven methods

Other types of data-driven methods exist and are succinctly introduced, to give a broader picture of the data-driven methods. They are however not tied to the work presented in the next chapters.

- **Correlation network**: Given the experimental data and the genes in presence, one can compute the pairwise gene correlations, which is directly reported as the weight of an undirected network. The particular interest is the scalability, as the complexity scales linearly with the number of experiments and quadratically with the number of genes.

- **Information Theoric scores**: intuitively, it consists in finding the Mutual Information (MI), which quantifies the degree of dependence of two random variables : the **MI** is zero when the two random variables are independent, and when they are deterministically linked, the **MI** returns the entropy of the marginal distribution. The formal equation is given here under:

$$MI[X,Y] = \sum_{x_i,y_j} P(x_i,y_j) log \frac{P(x_i,y_j)}{P(x_i)P(y_j)} = \sum_{x_i,y_j} P(x_i,y_j) log \frac{P(x_i|y_j)}{P(x_i)}$$

  where X and Y are the random variables, and $P(X,Y)$ their joint probability distribution. Applying this technique to GRN, the probability distributions to be used are the empirical distributions of the expression level for each pair of genes, estimated from the experiment samples. As the previous method, this straightforwardly gives a weight to each possible edge, ending up with a fully connected, weighted but undirected network. This method is better at finding complex regulatory relation, while having other drawbacks :

  1. the complexity is quadratic in both number of genes and samples,
  2. joint probability estimation is sensitive to noise in experimental data if the sample size is too small,
  3. non predictive.

1.3.5 Logical Models

One of the simplest and most basic idea in modeling methodology relies on the discretization of the data, and the logical thinking. Kauffman [36] and Thomas ([62, 64, 63]) introduced logic-based models applied to GRN in order to represent the state of each entity in the system.

[3] for which no data is available

As already stated, this entity is either a gene or any other gene product, or yet another element part of the gene regulatory network.

The state of each entity is represented as a discrete level, at any discrete time step. Often, the evolution over time is assumed synchronous : the levels of each entity are updated at each time step according to the regulation.

This type of model is discrete, reasoning mostly in qualitative terms, and abstracting the modeling from the continuous and noisy experimental data.

1.3.5.1 Boolean Network

Modeling a GRN in terms of a Boolean Network is introduced by Thomas in [62]. A Boolean Network is a dynamic model of time-discrete synchronous interactions between the nodes. A more detailed description can be found in [14] or [30].

As the mathematical framework introduced, a Boolean Network is a directed graph (X, E), where the nodes $x_i \in X$ are Boolean variables. The expression level is discretized into being above a limit, then "ON", `Boolean` value *true* or 1, or the expression level is below a limit, and the value of x_i is *false*, or 0. At any time, the state of the network is given by the values of all the nodes of the network: $State(t) = S(t) = (x_1(t), x_2(t), ..., x_n(t))$. To each node is associated a Boolean function, that takes as arguments the parent nodes of x_i in the network. That is, x_i is associated to $b_i(x_{i_1}, x_{i_2}, ..., x_{i_l}), l \leq n, x_{i_j} \in X$. A Boolean function is a function of Boolean variables connected only by logic operators. For each node, its Boolean function models the aggregated regulation effect of all its parent nodes, and is defined as

$$x_i(t+1) = b_i(x_{i_1}(t), x_{i_2}(t), ..., x_{i_l}(t))$$

In this model, the state of each individual nodes are updated synchronously, and all states'transitions together correspond to a *state* transition of the network from $S(t)$ to the new network state, $S(t+1)$. The passage form $S(t)$ to $S(t+1)$ is called a *trajectory*.

Although the logic underneath the model is an *oversimplification* of true biological behavior, as stated in [30], the Boolean networks have demonstrated their utility when reasoning over GRN:

- when the focus is on the generic principles rather than quantitative biochemical detail, Boolean networks can capture many biological phenomena, predict the trajectory between states, and provide a qualitative description of a system,
- the simplicity allows the model to be applied to larger GRN (containing a large number of nodes), when more detailed methods would be infeasible simply due to the lack of sufficient experimental data or computational complexity.

The simplicity of the method also brings drawbacks, as expressed in [30]:

- A steady-state of the continuous model based on differential equation (see Section 1.3.6 (p14)) will not necessarily be a steady state of the Boolean Model,
- Because of the intrinsic two-state nodes, the experimental data - measured on a continuous scale - need to be binarized. This step introduces risk and uncertainties in the modeling, as the key decision lays on the choice of the threshold between the two Boolean states,
- The Boolean Networks are deterministic, whilst true biological networks are known to have stochastic components. For instance, proteins can be produced from an activated promoter in short bursts that seem to occur at random time intervals.

An example of boolean network is given in Figure 1.7 (p13), presented in [55]. The GRN is represented in different forms : as (A) a graph, (B) rules, (C) complete table of possible states before and after transitions, and (D) as a state/transitions graph.

In the graph (A), the nodes **X**, **Y** and **Z** are connected by two kinds of edge depending if the source is an activator or an inhibitor. The symbol & implies that **X** and **Z** are expressed together. Based on this graph, the rules (B) can be expressed straightforwardly. In the truth table (C), the values for t are imposed and, based on the graph or the rules, values for $t+1$ are given. Considering the definition of a state given above and the truth table, the state transitions graph (D) can be drawn. The four representation are viewpoints of the same reality, expressed in different formalisms.

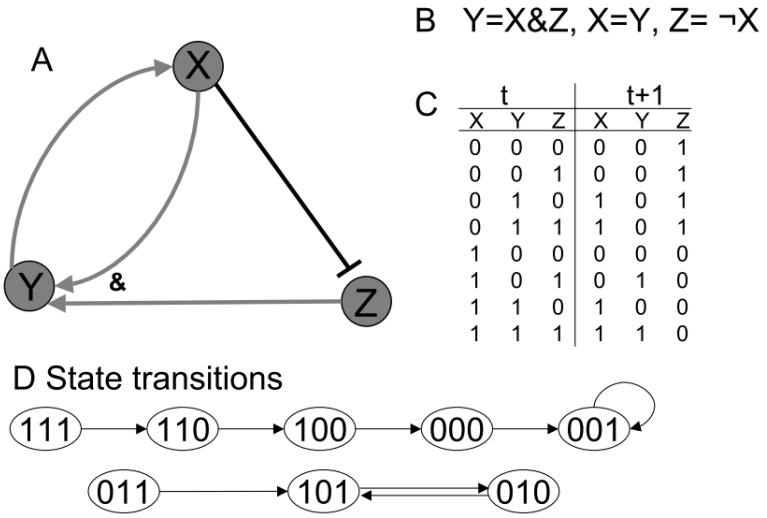

FIGURE 1.7: Example of a Boolean network

1.3.5.2 Beyond the boolean model

To go beyond some limitations of the boolean model, several improvements have been suggested by different authors:

- The asynchronous behavior of real system is discussed in [63], explaining the phenomena, and how to take it into account in the stable states definition and search,
- Multi-valued thresholds as explained in [50], to better fit to reality when required. It discretizes the concentration levels in several values, and as a result breaks the boolean principle ON/OFF, by introducing a threshold scale. This is further discussed in Chapter 3 (p33),
- Addition of the linearized equations ideas to the logical models, taking into account multi-valued principles, to the logical description, detailed in [51]. This is also further discussed in Chapter 3 (p33).

1.3.5.3 Probabilistic Boolean Networks

As indicated in [73], due to reasons such as the lack of experimental data, noisy data, or unknown assumptions over entities, it can be that several regulatory functions are possible for an entity. This raises the need to express uncertainty in the logic. The resulting model is a

boolean model where, at each time step, an entity is subjected to a regulatory function that is randomly selected among the pre-defined possibilities, according to pre-defined probabilities. The global amount of stable state can be considerably larger with this stochastic behavior introduction compared to the initial model.

1.3.5.4 Bayesian Networks

The graphical representation of a Bayesian network for GRN is a directed acyclic graph, where each node is a random variable representing a gene's expression, and the edges indicate dependencies, as reminded in [55] and [73]. Mathematically, the Bayesian Networks approach is Bayes' Theorem: for any two random variables X and Y, $P(X,Y) = P(X|Y)P(Y)$.

The random variables of the Bayesian network are drawn from conditional probability distributions : it implicitly encodes the *Markov Assumption* that "the future is independent of the past, given the present", [39, 83].

The details of the Bayesian networks methods are out of the scope of this present work - the interested reader is encouraged to consult [29] and [55] for a more thorough description.

1.3.6 Differential Equation Models and Linearization

A gene network can be modeled as a system of differential equations. It allows more detailed descriptions of the network dynamics by explicitly modeling the concentration changes over time.

The equations system relies on parameters that are usually difficult to measure or estimate accurately. Also, an important characteristic of the GRN is the robustness, as stated in Section 1.3.2 (p9). If the equations system does not converge to a solution (is not stable) when the parameters slightly change, it is doubtful that the system represents a real GRN. Hence, the real networks are robust with respect to variations in the substances' concentration, and the corresponding system should keep this stability property. Wehn the system is stable, from a mathematical point of view, the exact values of its parameters should not be critical. Several models exists and are introduced in [29] or [26].

Many challenges remain in the computational efficiency (as for many modeling techniques), as well as in the accuracy of the equations system itself, including the parameters value.

1.4 Practical case : the infamous lac operon

One of the most well-known, and well studied, real-life example of a transcriptional control is found in *Escherichia Coli*, also called ***lac* operon**. The interested user can watch the MIT given class on the topic in [37]. Abundant other literature references exist and are easily found on the web. This section uses some biological vocabulary (the names of proteins, genes, sugar, ...) that is not itself relevant for the understanding of the thesis.

1.4.1 Overview of the lac operon

The bacteria E. Coli, in order to grow, needs sugar. Its favorite sugar is glucose, a monosaccharide. When the glucose levels are low in the environment, the E. Coli needs to switch to alternative sources of energy (other types of sugar) to continue growing. Another possible source of sugar is the lactose, which is a disaccharide. Without going into details in this thesis, the lactose is composed by two monosaccharides that are glucose and galactose. In order to metabolize the lactose, the bond between the two monosaccharides must be broken. Of course, if glucose

Glucose	Lactose	Z, Y, A transcription
0	0	−
0	1	+
1	0	−
1	1	−

TABLE 1.1: Transcription of the three structural genes based on the presence of sugar

is available, there is no need to synthesize a protein that will cleave lactose into glucose and galactose.

The lac operon consists on three genes involved in the processing of the sugar lactose – a disaccharide. The three genes are:

Z that codes for a protein called β-galactosidase. This protein is responsible of splitting lactose into glucose and galactose. It is in the heart of the lac operon system.

Y that codes for a protein called permease

A that codes for a protein called transacetylase

As they are part of an operon, Z, Y and A are grouped together : they have a single controlling region.

A first logical table can be established, where "1" means "present" and "0" means "absent" for glucose and lactose, and "+" or "-" denotes an increase of decrease of the genes Z, Y, A products concentration levels (directly related to their transcription).

The results in Table 1.1 (p15) for the transcription are well understood when no sugar is present, or when one of the sugar is present. However, the transcription result when both glucose and lactose are present seems odd, as one could imagine that E. Coli would simply pick the preferred sugar, and the transcription could go on. The result is explained in the following sections.

1.4.1.1 DNA structure of the lac-operon

The DNA structure of the lac-operon is given in Figure 1.8 (p16). As indicated previously, the operon is made of the three structural genes Z, Y and A. Upfront this triplet is found the operator site (o), part of the regulatory sequences, which is composed by a promoter for regulatory gene (p_i), the regulatory gene (i), and the promoter for the operon, p_{lac}. p_i and i are not per se part of the lac operon, but they play a key role in the regulation hence are part of the regulatory sequences for the structural genes Z, Y and A.

This DNA structure helps understanding how the lactose can be properly metabolized when needed, according to Table 1.1 (p15).

1.4.2 Different situations

1.4.2.1 Lactose is absent

When there is no *lactose* in the environment, the **repressor** protein *LacI* is synthesized continuously by the gene i. This protein will bind with the promoter site, right in front of the operon, preventing the transcription of the operon. No β-galactosidase or other A,Y,Z products are generated as imaged in Figure 1.9 (p16).

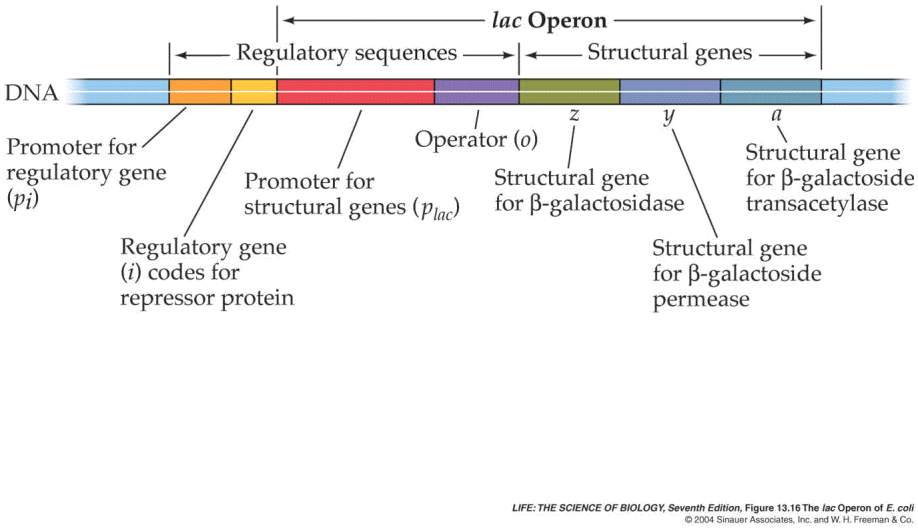

FIGURE 1.8: DNA Structure of the lac-operon

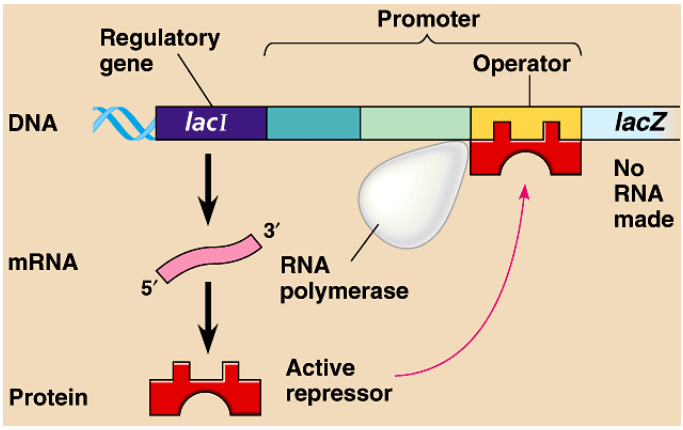

FIGURE 1.9: No lactose, repressor lacI is transcribed and blocks the A,Y,Z transcription

1.4.2.2 Lactose is present

When *lactose* is present in the environment, the sugar lactose (a dissacharide) fits onto the repressor protein, at a specific active site. As a result, the repressor protein will change its shape, in such a way it can no longer bind the operator site. The mRNA can reach the structural genes and the Z,Y and A products can be synthesized.
As the β-galactosidase is produced when lactose is present, the lactose acts as an inducer for the enzymes responsible of its metabolization.

Though, it still does not explain why the operon is not transcribed when lactose and glucose are present together: another mechanism necessarily needs to be set up to prevent the operon transcription when both sugar are present in the environment. This is explained in the following sections.

Glucose	Lactose	CAP-cAMP	lacI	Z, Y, A transcription
0	0	+	⊕	−
0	1	+	⊖	+
1	0	⊖	+	−
1	1	⊖	−	−

TABLE 1.2: Transcription of the three structural genes based on the presence of sugar

1.4.2.3 Control of the lac-operon expression in presence of Glucose

Two other proteins help understanding the process in which the presence or absence of glucose controls the transcription of structural gene through the control of the RNA polymerase binding:

- CAP, that stands for Catabolite activator protein,
- cAMP, that stands for cyclic Adenosine MonoPhosphate.

Together, they form the CAP-cAMP complex, which binds DNA. When CAP-cAMP binds DNA, the efficiency of RNA polymerase binding is increased at the lac operon promoter, which increases the transcription of structural genes.

$$\begin{aligned} \text{CAP-cAMP on DNA} &\xrightarrow{\text{reinforces}} \text{RNA polymerase on lac operon promoter} \\ &\xrightarrow{\text{increases}} \text{Z,Y,A transcription} \end{aligned}$$

The formation of CAP-cAMP complex is directly linked to the presence or absence of glucose in the environment : when the level of glucose is low, the concentration cAMP is abundant and the complex CAP-cAMP is formed; when the level of glucose is high, the CAP-cAMP complex does not form, and the RNA polymerase cannot bind to the lac-operon promoter efficiently.

Glucose and Lactose are present. As the glucose is present, the complex CAP-cAMP does not form. The RNA polymerase cannot really bind to the promoter, and the structural genes are not transcribed.

Lactose only is present. As the glucose is absent, the complex CAP-cAMP is abundant, allowing the binding of the RNA polymerase on the promoter site. The structural genes can be produced.

1.4.2.4 Summary of the evolution

Table 1.1 (p15) can be explained thanks to two other entities : the product of the gene i and the complex CAP-cAMP.

Considering the evolution of the concentration levels with a + denoting an increase, and a − denoting a decrease, the original table can be completed with the two regulatory entities. The result is given in Table 1.2 (p17). The values circled are the leading effect of each line.

1.5 Conclusion

This chapter succinctly introduces the reader with the biological notions required to understand gene regulatory networks and their utility. It gives as well some references to the current main techniques used to infer or model the GRN.

Those techniques all experience the difficulty to wisely give a value to the parameters of the model. Furthermore, the GRN are not by themselves the final outcome of the inference, but only an intermediate result. As suggested in [10], tools such as platforms should exist to serve the biologist or medical expert: it would allow the downstream analysis of gene networks.

Such analysis and visualization tools would benefit from being accessible to non-technical experts, and help intuitively reason on those gene regulatory networks. A usable tool should enable not only the inference of GRN's, but also, more importantly, to evaluate hypotheses on known and unknown networks, to confront the experimental data allowing to vary parameters, to integrate data uncertainties, etc.

This thesis introduces the reader to a tool concept built in that purpose: to give to a non-technical expert the opportunity to infer a GRN based on data, and to assess its hypotheses or reason on an existing network, based on her/his inputs.

Chapter 2

Constraint Logic Programming

2.1 Introduction

This chapter familiarizes the reader with the main computer science topic of this thesis: the constraint logic programming paradigm.

Firstly, an overview describes the constraint programming paradigm, and introduces the notions that come up in this work. Topics as constraint propagation, search – specifically backtracking –, problem modeling and labeling are largely addressed.

Secondly, having presented the required notions, the paradigm of constraint logic programming is introduced. A simple yet illustrative example of the logic programming paradigm is given, then some tools that can be used in the context of constraint logic programs are presented. In particular, Prolog, the computer language which is abundantly used in this thesis, is introduced. An example of usual problem – the N-queen problem –, and its resolution using the constraint logic programming is given, followed by the presentation of the nominal structure of a resolution.

2.2 Constraint Programming

Constraint Logic Programming is a powerful computing paradigm particularly indicated to solve combinatorial search problems, as presented in [52].

The main idea is rather straightforward : the user states the constraints, and a *general purpose* constraint solver is used to *solve* the constraints.

The *constraints* are relations, and a *constraint satisfaction problem (CSP)* consists in a set of variables, each associated with some domain of values, and a set of relations – the constraints – on subsets of these variables. A more formal definition can be found in [53] and [22].

Constraint solvers take a real-world problem represented, according to a CSP, as decision variables and constraints, and will look for an assignment of all the variables that satisfies the constraints. This assignment can be extended for instance so that the solution is optimized according to specific criteria, or such that all solutions for which the constraints are fulfilled are found by the solver.

The constraint solvers search the *solution space* systematically, using different techniques such as backtracking (see Section 2.2.3 (p22)), branch and bound[1], ... Those systematic methods mix the search part with an inference part, where the information contained in a constraint is propagated throughout the other constraints (see Section 2.2.2 (p20)). The goal of this inference is to reduce the search space.

[1] Other techniques as local search exist. This is out of the scope of this introduction.

Some constraints, called *global constraints*, apply on sequences of variables: they generally represent real-life constraints, and come with special propagation procedures. They often allow the systematic search to be more efficient and more effective, being optimized in the solver's implementation.

The main task, and probably the most complicated one, consists in the *good* modeling of the real-world problem into constraints : defining the model that works well with a chosen solver is not easy. Among the main difficulties are the symmetry of the solutions to a problem, or the fact that many real-world problems are over-constrained. Specific techniques involving the addition of *soft constraints* exits, as well as technique to guide to an optimal solution.

Generally speaking, a constraint solver can be implemented in any language, while some are particularly indicated as they already rely on similar notions: logic-based programming languages, such as Prolog, that natively rely on relations and backtracking search.

2.2.1 Classification

2.2.1.1 Domains

There is theoretically no limitation of the possible domains on which the constraint programming paradigm can be applied. As presented in [22], a basic CSP involves variables that are *discrete* and have *finite domains*. In this case, if the domain size of any variable is d, and the number of variables in n, the number of possible complete assignments is $O(d^n)$, exponential in the number of variables. A specific case is when the domain is limited to boolean values *true* or *false*, for which several specific and optimized library are built (see [58] for instance, as a specific Prolog Boolean CLP library).

Discrete variables can have *infinite* domains, as integers or naturals, for instance. Enumerating all possible combinations is no longer possible, and specific solution algorithms must be used, such as the Simplex algorithm, for linear constraints over infinite domains, introduced in [7] or [69].

In real world applications, several problems require *continuous* domains. A specific and well-known category of such problems is the *linear programming* problems : the constraints are only linear (in)equalities. They can be solved in a time polynomial in the number of variables. Other category (quadratic constraint programming, ...) are still being studied.

2.2.1.2 Constraints

Different types of constraints exist: unary - involving one variable, binary - between two variables, or high-order. Those constraints may be absolute: any violation of such a constraint does not lead to a solution.

In the real-world, the solution often results from trade-offs and preferences. A solution is needed, even if all conditions are not fulfilled. This leads to another type of constraints being the *soft* constraints, as discussed in [22]. These constraints allow an improved optimization and cost function : the respect of a soft constraint will decrease the cost of the solution, while a violation of a soft constraint - although still being part of the solution - increases its cost. The optimal solution is the one having the lowest cost.

2.2.2 Constraint Propagation

The notion of constraint propagation is linked to the notion of consistency. As defined in [52], a *local inconsistency* is an instantiation of some of the variables that satisfies the relevant constraints, but cannot be extended to one or more additional variables. This partial instantiation will not end up to a global solution, and should therefore be rejected. Using the backtracking

search to find a solution, this type of inconsistencies can be the reason of numerous dead-ends, implying useless effort. Effective constraint propagation techniques have emerged in order to avoid losing computational power in infertile work.

To understand the propagation and related consistencies, a CSP is to be seen as a graph, as explained in [53] : the nodes are the variables, and the arcs, or edges, are the constraints. More generally, an hypergraph[2] represents a CSP such that the nodes are connected by hyperedges, as shown on the example below and Figure 2.1 (p21).

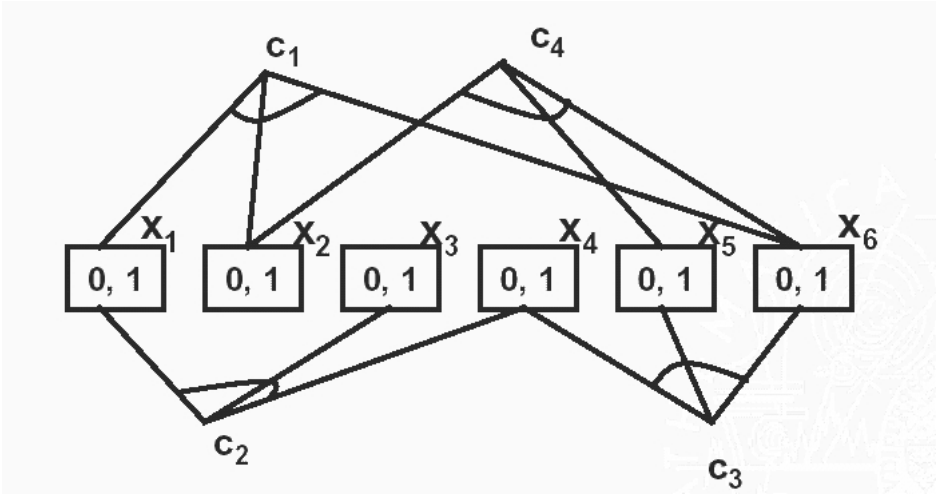

FIGURE 2.1: Hypergraph representation of a CSP

```
 Example
---------
variables: X1, X2, X3, X4, X5, X6 with domain 0..1
constraints:
c1: X1 + X2 + X6 = 1
c2: X1 - X3 + X4 = 1
c3: X4 + X5 - X6 > 0
c4: X2 + X5 - X6 = 0
```

2.2.2.1 Local consistency

Different local consistencies exist :

> **Node consistency**: each value from the domain of variable V_i satisfies all unary constraints over V_i
>
> **Arc consistency**: arc(V_i, V_j) is arc consistent, iff there is a value y for each value x from the domain of V_i such that the assignment $V_i = x, V_j = y$ satisfies all binary constraints over V_i, V_j. The arc consistency is directional: consistency of arc(V_i, V_j) does not guarantee consistency of arc(V_j, V_i).
> CSP is arc consistent, iff all arcs, in both directions, are arc consistent.

Arc consistency is an important local consistency technique in practice. A given constraint can be made arc consistent by repeatedly removing the values that lead to an inconsistency. Removing such values is called *pruning* the domains. Enforcing arc consistency over the

[2]generalization of a graph in which an edge can join any number of nodes.

complete CSP requires to iterate over the domains until a fix point is reached, where all arcs are consistent. Although efficient algorithm are developed, it could represent a heavy load: a trade-off exists between the cost of the propagation, and the amount of pruning. A common technique to reduce the propagation costs is *bounds consistency*, where only the extrema values of a domain must be consistent (must have a value for the other variable of the constraint such that it is satisfied). Bounds consistency is weaker than arc consistency, but has proven to be useful, specifically for arithmetic constraints and global constraints.

The node and arc consistency can be generalized to *k-consistency*, as presented in several sources such as [53].

2.2.2.2 Global constraints

Intuitively, a global constraint is a constraint over a sequence of variables. Their interest relies often in the accompanying pruning strategies that come with them, which are largely optimized in the solvers. Expressing global constraints allows reducing the search space efficiently, hence help prune the search tree.

One of the most speaking examples is the `all_different` constraint. `all_different`, applied to a sequence or a set of variables, forces all these variables to be pairwise different. This corresponds to many real-world situations.

Most of the global constraints are built-in in constraint solving systems. Their benefits is double: firstly, it eases the modeling task of the programmer, and secondly it prunes more efficiently the search space. Example of the complexity is reported in [52] or [24] Among the most famous global constraints are, despite `all_different`,

`gcc`, that states that over a set of variables and values, the number of variables instantiating to a value must be between two bounds (which can be different for each value),

`cumulative`, which is specifically designed to handle task/resources/time problems.

2.2.3 Backtracking Search

The search is probably the most algorithmic technique used for solving CSP. Considering a finite search tree, it can either be complete or incomplete. Complete search guarantees that a solution will be found if it exists, or can be used to prove that no solution to the CSP exists. On the other hand, incomplete search cannot be used to prove that no solution exists or to find a provable optimal solution, but is often an effective way to find a solution if one exists, and an approximation of an optimal solution. In the scope of this work, we will mostly focus on a complete search technique, the backtracking[3].

The backtracking search strategy is a depth-first traversal of a search tree, which is generated as the search progresses. The overall idea is to select, at each node, a variable that is uninstantiated, and extend the node with different branches out representing the alternatives, or choices, that may have to be examined to eventually find a solution. Several techniques exist to improve the efficiency of a backtracking search algorithm, among which the constraint propagation, nogood recording, heuristics for variable and value ordering.

2.2.3.1 Constraint propagation during search

One method to improve the efficiency of the search is to maintain the level of local consistency (see Section 2.2.2.1 (p21)) during the backtracking, and to perform the constraint propagation at each node of the search tree. The main benefit is the pruning the search tree:

[3] As expressed, complete only if the search tree is finite

- by removing a complete branch without solution due to an empty domain for a variable,
- by reducing the domain of a variable to a single value, that does not require further branching,
- by reducing the domain : fewer further branching

Rossi, in [52], refer different authors that suggested constraint propagation techniques. Usually, those techniques are built-in in the constraint solvers.

2.2.3.2 Nogood recording

Nogood recording consists in adding implied constraints, also called nogoods. A constraint is considered implied if the set of solutions to the CSP is the same, with or without the constraint. The goal of adding such implied constraints is to:

- Remove many dead-ends from the search tree,
- Help discovering other dead-ends easily (in an easier way than without the implied constraint)

Rossi, in [52], suggests three techniques:

1. Add implied constraints by hand during modeling phase,
2. Add implied constraints automatically by applying a constraint propagation algorithm,
3. Add implied constraints automatically after a local inconsistency or dead-end is encountered

The overall idea is that the added constraints will prune the search space in the future, reducing the size of the problem.

More information on nogoods can be found in [35] and [52].

2.2.4 Modeling

As previously stated, the overall idea of constraint programming is straightforward: the user states the constraints, and a *general purpose* constraint solver is used to *solve* the constraints. Stating the constraints of the problem refers to modeling the problem. It remains one of the most important challenges of the methods: problems usually can be modeled in different ways, logically equivalent, while the efficiency of the solver can have different outputs. Furthermore, a modeling efficient for one solver is not de facto efficient on other solvers. Properly modeling a problem is a matter of the programmer's experience, although several good practices have been identified:

- Decide if constraint programming is an appropriate technology. A possible hint is to extract from the real-world problems or the end-user the constraints and try to build a model.
- Define the variables, their domain, and the constraints that apply. Constraints should be expressed easily,
- Symmetry. Naturally, many real-world problems present symmetries in their answers, that often are irrelevant for the end-user. Removing this symmetry - pruning the search space - can drastically help to reduce the search time, while keeping the most important solution variations. In order to remove symmetry, several techniques can be applied :

- Adding constraints on the variables to eliminate the symmetric solutions,
- Posting simple precedence constraints on the values, if they are originally interchangeable. The example introduced in Section 2.3.4 (p29) uses this technique,
- Modify the search procedure (see Section 2.2.3 (p22)) to avoid visiting nodes that lead to symmetric solutions.

2.2.5 Labeling

Once the variables are defined, and the constraints are modeled, the remaining task is to assign, to each variable, a value compatible with all the constraints. This step is usually called the labeling. As already introduced in Section 2.2.3 (p22), it consists in two sub-steps:

1. Decide an effective ordering to label the variables,
2. Decide which value from the domain will be assigned first

2.2.5.1 Variable and value ordering heuristics

This sequence of decisions, called variable and value ordering, can have a major impact on the performances of the (backtracking) search. A simple strategy can be to label the variables in the simple order given by the context, then try the value starting from the lower domain bound (provided it is a finite domain) to the upper bound. However, many alternative heuristics are based on choosing the variable with the smallest number of values remaining in the domain. Depending on the particular situation and real-world problem to solve, the heuristic can be modified to improve the performances.

A famous example of a variable ordering heuristic is called the *first fail*, usually considered as an efficient technique. A *ff* labeling selects the remaining variable with the smallest domain to label first. It follows the idea: *"To succeed, try out first where you are most likely to fail"*. An important consideration when selecting values and variables ordering is that it depends on the modeling part of the problem. As stated in the Section 2.2.4 (p23), different models can represent real-life problems. Specific labeling techniques may be more adapted to a particular modeling. Empirical comparison on a subset of inputs may be worthwhile in order to observe the difference in performances, and select case by case the most appropriate heuristic.

2.3 Constraint Logic Programming

The Constraint Programming paradigm can be embedded in many environments, but several are more suited or convenient, as relying on the same reasoning. In particular, the following characteristics make constraint paradigm naturally close to logic programming.

- constraints can be seen as relations - or predicates -,
- their conjunction corresponds to a logical *and*,
- backtracking search is the basic search algorithm to solve constraints,

Together, they form the Constraint Logic Programming paradigm, which semantics can be found in [33].

2.3.1 Logic Programming

The logic programs are logical implications between collections of predicates.

The program is constituted by : a set of rules, the *clauses*, which relate the truth value of a literal, the *head* of the clause, to the collection of other literals, the *body* of the clause. Executing such a program looks for the truth value of a statement. This statement is called the *goal*. Repeatedly, the goal is transformed during the resolution steps until the goal is empty - and the proof is successful - or the goal is not empty but no extra resolution step is possible - this is a failure, the goal is false - or the resolution continues forever - infinite computation.

Each resolution step involves a unification between a part of a goal, and the head of a clause.

An example of such a program is given hereunder. The first thing to notice are the facts, that state what is true. The predicates `female/1` and `male/1` indicate literals representing a (fe)male person. `parent/2` indicates who is the parent of whom, such as in `parent(X,Y)`, `X` is the parent of `Y`. The second part of the program consists of the rules, as defined above. For instance, `father(X,Y):- parent(X,Y), male(X).` states that if the conjunction of `parent(X,Y)` and `male(X)` holds, than `father(X,Y)` holds, and vice versa. Intuitively, it is simple to state that `X` is the father of `Y` if `X` is the parent of `Y` and `X` is a male.

Theoretically, there is no ordering in the way the predicates are inserted. In concrete implementation of the logic program interpreters, however, proper ordering allows to improve the efficiency of the resolution, and to avoid infinite loop. As a simple example, the base case of a recursive predicate should be placed before the recursive case.

```
% Family - Logic Programming - Prolog (SWI-Prolog version 7.6.4)
% Facts
        male(geoffroy).
        male(etienne).
        male(olivier).
        male(valery).
        male(jacques).

        female(vero).
        female(laetitia).
        female(camille).
        female(suzane).
        female(madeleine).

        parent(etienne,geoffroy).
        parent(etienne,camille).
        parent(etienne,laetitia).
        parent(vero,geoffroy).
        parent(vero,camille).
        parent(vero,laetitia).
        parent(suzane,vero).
        parent(jacques,vero).
        parent(valery,etienne).
        parent(madeleine, etienne).

% Rules
        father(X,Y):- parent(X,Y), male(X).
        mother(X,Y):- parent(X,Y), female(X).
        grandparent(X,Z):- parent(X,Y), parent(Y,Z).
        offspring(X,Y) :- parent(Y,X).
        sister(X,Y):- parent(Z,X), parent(Z,Y), female(X), X\=Y.
```

```
brother(X,Y) :- parent(Z,X), parent(Z,Y), male(X), X\=Y.
```

Asking for the true value of a goal can be done in several ways - or serve different purposes. Several executions are shown below.

2.3.1.1 male(geoffroy).

The most simple resolution of this program consists in asking if the goal `male(geoffroy).` is true.

```
?- male(geoffroy).
true.
```

The interpreter answers `true`, as `male(_)` can be unified with a fact stated in the program.

2.3.1.2 male(X).

A more interesting goal is to ask who is a male. For that purpose, the goal is `male(X)`, where `X` is a variable.

```
?- male(X).
X = geoffroy
```

The answer is `X=geoffroy`, meaning that the goal is true provided that `X` is unified with `geoffroy`. If one presses on the ";", other solutions are shown:

```
?- male(X).
X = geoffroy ;
X = etienne ;
X = olivier ;
X = valery ;
X = jacques.
```

Indeed, in the knowledge base – the program –, `male(X)` is `true` if `X` is either `geoffroy` or `etienne` or `olivier` or ...

This shows one of the main properties of logic programming: the ability to show and find all solutions.

2.3.1.3 sister(laetitia,Y).

This last example of execution shows a slightly more complex predicate. Intuitively, again, we would like to know who is `laetitia` the sister of. As we know the family, the answers are clearly `camille` and `geoffroy`. The predicates `sister/2` states that `sister(X,Y)` is *true* when `X` and `Y` have the same parent, `X` is a female, and `X` is different than `Y`[4].

Asking the interpreter, the execution gives all the solutions, using the ";".

```
?- sister(laetitia,Y).
Y = geoffroy ;
Y = camille ;
Y = geoffroy ;
Y = camille ;
false.
```

[4]Semantically, one could argue that the `parent/2` predicate is not enough has a *half-sister* would also be unified and considered as sister. This discussion is not relevant in the context of logic programs

This execution enlightens one of the main features of logic program resolution, the backtracking: after `geoffroy` and `camille` are discovered, the system "backtracks" to the previous choice point, where it chooses the alternative as asked by the ";" command. In this specific case, the second path of resolution gives the same answer.

The backtracking can be used in two different occasions: when looking for other solutions, as done here with the ";", or in case a decision made at a choice point did not lead to a `true` value of the goal. The system automatically backtracks to the last choice point not explored yet in order to try a different resolution path.

More information on the logic programs can be found in [32].

2.3.2 Constraints logic programs

Adding the constraint paradigm to the logic paradigm syntactically consists in considering constraints as special predicates. Semantically, as introduced in [52] or in [33], several aspects are improved:

- The concept of unification is extended to constraint solving, and a goal can therefore be described via some term equations but also general constraints.
- It improves in many cases the resolution, adding the arithmetic reasoning directly on top of the logic semantics of the Herbrand universe.
- The constraint solver allows the combination of the inherent backtracking search of logic programming with the constraint propagation (see Section 2.2.2 (p20)), improving the overall efficiency.

CLP can be applied to various classes of constraints. Choosing a particular language induces the choice of a certain constraints class, and an adequate constraint solver. For instance, CLP(FD) stands for Constraints Logic Programming over Finite Domain, CLP(B) stands for CLP over Boolean Domain, etc.

2.3.3 Tool for Constraint logic programming

CLP naturally blends in logic programming tools. At the University of Namur, Prolog is the commonly used logic programming language.

Origin of Prolog. As related in [82], founded in 1972 in Marseille by Colmerauer and Roussel, Prolog is designed to reconcile the use of logic as a *declarative* knowledge representation language with *procedural* knowledge representation. Originally, the intent was to work on natural language processing (computer and human language interactions), but the scope has been broadened ever since.

Currently, Prolog remains largely used in the context of logic programming for expert systems, artificial intelligence, theorem proving and - as originally - natural language processing. Since 1995, Prolog has an ISO/IEC standard : the ISO 13211. The last correction is dated 2017.

Different implementations. Prolog has different implementations. The list is too wide and represents limited interest to be elicited in the present work. A core using purely ISO-Prolog language is supposedly portable from an implementation to the other. Unfortunately, it is far for covering all the needs: many functionalities are not regulated by the standard, leading to non portability caused by multiple factors such as :

- Bounded or Unbounded integer arithmetic,
- Additional types (not covered by ISO),
- Multi-threading,
- Use of libraries unavailable in other implementations.

In the context of this work, several elements were required or wished to select the implementation, among which:

1. Free license,
2. ISO-Prolog syntax,
3. Support of Constraint Logic Programming,
4. Support of HTTP (see Section 4.3 (p42)),

The SWI-Prolog implementation ([59]), fulfills all these requirements. What's more, SWI-Prolog is mature (from 1987), still active (last release in January 2018), and provides a solid documentation online. For these reasons, SWI-Prolog is the Prolog implementation that will be actively used in the context of this work.

CLP Extension. The CLP extension of core Prolog is declined in different constraint solvers for the different domains, or constraints classes (see Section 2.3.2 (p27)).

SWI-Prolog provides four constraints solvers, to solve constraints over Boolean variables (`clpb`), integer variables (`clpfd`), rational numbers (`clpq`) or floating point numbers (`clpr`).

Due to the main context of this work, the library used will be `clpfd`, as all the parameters and data can be discretized. `clpfd` is originally developed by Markus Triska, see [68]. Many online examples are also available in [67].

Other tools. Other tools exist, which are usually used as a library or a complete interface for the user, abstracting the solving tasks. Choco and Minizinc are among the most popular of those possibilities.

- Choco, presented in [48], is a Free Open-Source Java library dedicated to Constraint Programming.
- Minizinc is a free and open-source constraint modeling language. The real-world problem is written in a high-level language, solver-independent, and is then compiled into FlatZinc, a solver input language that is understood by a wide range of solvers, including Choco, ECLiPSe, SICStus Prolog, etc. The complete list of compatible solvers is given in the website, [40]. This sounds promising, as the modeling is not linked or blocked by a solver, but an abstraction layer exists.

Although that would be interesting, broadening the scope of the current work using those solvers/tools is left for future work.

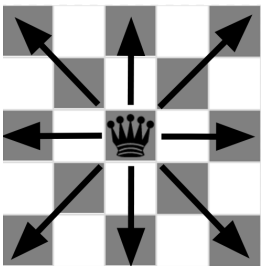

FIGURE 2.2: Chess queen acceptable movements

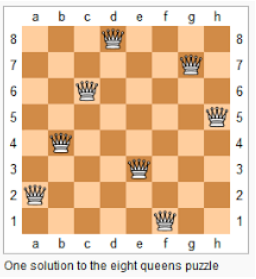

FIGURE 2.3: 8-queen solution

2.3.4 N-Queen - Example of CLP(FD) resolution

To sum up the theoretical framework explained in this chapter, a concrete example of problems resolved using constraints logic programming is given and detailed in this section.

The infamous N-Queen problem is detailed as well in [16]. The problem can be phrased as placing N chess queens on an $N \times N$ chessboard so that no two queens attack each other. Figure 2.2 (p29) gives an overlook of the acceptable movements for the chess queen, and Figure 2.3 (p29) shows a solution for a 8×8 board.

Several techniques can be employed in order to solve this problem: for instance, one could think of generating and testing one by one the different alternatives until a proper solution is discovered. For small values of N, it could be acceptable. As soon as the number of queens increases, the amount of possibilities exponentially increases as well, leading to a practically unsolvable problem. In general, there are N^N different possibilities to explore: table 2.2 (p29) gives an idea of this number for several cases.

N	Number of configurations
1	1
2	4
4	256
8	16777216
16	$1,844674407 \times 10^{19}$

TABLE 2.2: Number of configurations to test

Another technique is to define the N-queen problem as a constraint satisfaction problem. This CSP can be defined as a 3-tuples where:

- $Q = \{q_1, ..., q_N\}$, a set of variables. q_i represents the queen in the column i. Associating a queen to a column - as a rapid understanding of the problem suggests - is already a pruning of the possible search space.
- $D = \{D_1, ..., D_N\}$, a set of domains, one for each of the variables. Each domain is finite, and contains the possible values for the corresponding variable $\{1, ..., N\}$,
- $C = \{C_1, ..., C_c\}$, a set of constraints. As a reminder, a constraint is a relation defined on a subset of Q.

The goal is to find a value for all the variables $\{q_1, ..., q_N\}$ that satisfies all the constraints $\{C_1, ..., C_c\}$. The problem description and understanding suggests the conditions that need to be fulfilled: two queens cannot attack each other. Consider q_i and q_j, with $i \neq j$, being two queens. The values denoted by the variables are the rows the queens are placed in. The constraints can therefore be established.

C1 not in the same column. This is verified per construction,

C2 not on the same row: $q_i \neq q_j$,

C3 not along the same diagonal $|i - j| \neq |q_i - q_j|$.

And the solution to the problem will be *any* assignment of values to the variables $q_1, ..., q_N$ that satisfies all the constraints.

Numerous implementations can be found on the web. Triska's implementation using Prolog and its CLP(FD) library is available in [67].

The resolution can be detailed and visualized for the case $N = 4$, as shown is Figures 2.4 (p31), 2.5 (p31), 2.6 (p31) and 2.7 (p31). After modeling the constrained as described above, the labeling phase starts assigning an acceptable domain value to the variables q_1, q_2, q_3 and q_4. The domain of q_i is at that time $[1, 2, 3, 4]$. Considering the assignment starts from the leftmost variable, and the minimum value of the domain, the system first tries to assign the value 1 to q_1. Applying consistency techniques as presented in Section 2.2.2 (p20), the acceptable domains for other variables are reduced, hence the reddish color in the left chessboard of Figure 2.4 (p31).

$$\begin{aligned} q_1 \to 1 \quad &\Rightarrow \quad q_2 \in [3, 4] \\ &\Rightarrow \quad q_3 \in [2, 4] \\ &\Rightarrow \quad q_4 \in [2, 3] \end{aligned}$$

The system then assigns an acceptable value in the domain of q_2, the first one being 3. Propagating the constraint, it leads effectively to no solution, as the domain of q_3 would not contain any acceptable value anymore, as shown in Figure 2.4 (p31). Applying the backtracking search technique, the systems goes back to the last choice performed, *rewinds* its last assignment $q_2 \to 3$ and removes the propagation linked to this assignment. A new choice is performed, as represented in Figure 2.5 (p31): q_2 is assigned to 4. Applying the propagation for this new assignment, the domains for remaining variables are given by:

$$\begin{aligned} q_1 \to 1, q_2 \to 4 \quad &\Rightarrow \quad q_3 \in [2] \\ &\Rightarrow \quad q_4 \in [3] \end{aligned}$$

Pursuing the labeling with the third variable q_3 leads also to a dead-end, as there is, after application of the consistency, no more acceptable value in the domain of q_4.

At this point, the system needs to backtrack to the previous choice point: up to the assignment of q_1. Instead of assigning $q_1 \to 1$, the system tries the second acceptable value of the domain, considering all other propagation applied earlier are removed – when the system backtracks, it needs to let down the consistency checks performed with previous choices. The domain of q_1 is therefore $[1, 2, 3, 4]$ and the value 1 has already been tested and lead to no solution. q_1 is then assigned to 2. The situation is represented in Figure 2.6 (p31). In the exact same way as previously explauned, the consistency techniques are successively applied, for the four variables. With this assignment, the result is different: all variables can be assigned with an acceptable value of their domain. That is, a solution exists, hence the green color on the chessboard depicted.

This is the first solution of the problem.

If another solution is desired – if the user presses ";" for instance –, then the system backtracks again, up to the choice point of the q_1 assignment. This choice can be removed and the system can try out $q_1 \to 3$. The situation is shown in Figure 2.7 (p31). An experienced

eye directly recognizes the situation the system is in: a new solution is provided with the assignment, per symmetry with respect to previous assignment.

Finally, backtracking to try the assignment $q_1 \rightarrow 4$ does not lead to any solution, as this is the symmetrical problem with respect to $q_1 \rightarrow 1$.

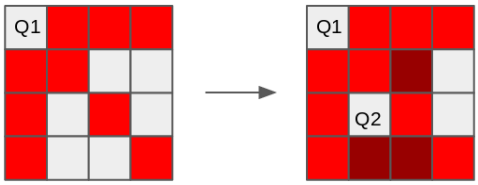

FIGURE 2.4: 4-queen, No solution if $q_1 = 1$ and $q_2 = 3$

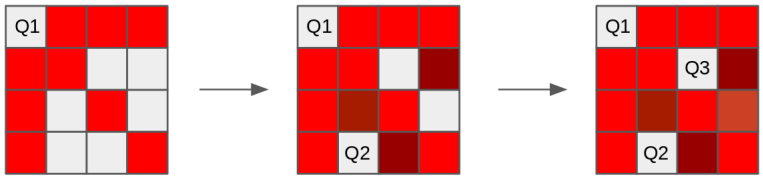

FIGURE 2.5: 4-queen, No solution if $q_1 = 1$ and $q_2 = 4$

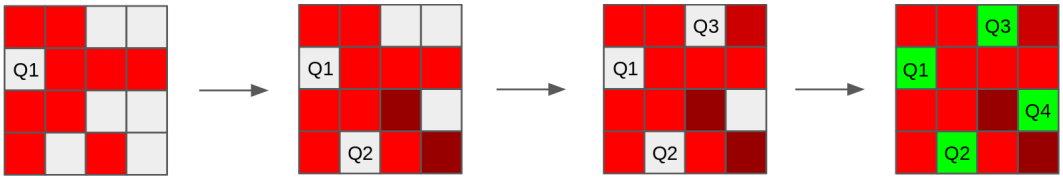

FIGURE 2.6: 4-queen, First acceptable solution

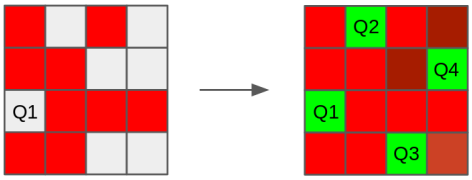

FIGURE 2.7: 4-queen, Symmetric acceptable solution

The execution of the program written by Triska in [67] confirms the solutions found above.

```
?- n_queens(4,[Q1,Q2,Q3,Q4]), labeling([],[Q1,Q2,Q3,Q4]).
Q1 = 2,
Q2 = 4,
Q3 = 1,
Q4 = 3 ;
Q1 = 3,
Q2 = 1,
Q3 = 4,
Q4 = 2 ;
false.
```

2.3.5 General Resolution of CSP using CLP

A problem modeled using constraints, a Constraint Satisfaction Problem (CSP), can be solved using Constraint Logic Programming by a nominal canvas. It consists of three steps in order to take usually the most out of the constraint solver, and guide the programmer. Of course, experienced programmers may want to revisit this canvas to adapt it to a specific situation. This is not covered in this work.

The three parts are the variable definition and domain specification, the constraints elicitation, and the labeling.

2.3.5.1 Variable definition and domain specification

Considered as the very beginning of the modeling part, it consists in choosing, from the real world problem, what are the variables on which constraints will need to be applied. Ideally, for all these variables, a domain should be specified.

Many real world problems can have a simpler representation (inducing finite domains) when abstracting details or limiting redundant solutions. For instance, the N-Queen problem limits redundant solutions by directly imposing that a queen belongs to a specific column. The only remaining variables are therefore the rows. This is a smart choice of variables.

2.3.5.2 Constraints Elicitation

Once the variables are known, the constraints need to be elicited. In particular, real world problems often have global constraints. Soft constraints can be introduced. If possible, implied constraints and symmetry-breaking constraints can be introduced as well, in order to limit the number of possible solutions while keeping all the relevant ones.

2.3.5.3 Labeling

When the variables are known and the constraints are applied, the solver can look for a solution, assigning a specific value successively to all the variables. As introduced in this chapter, *successively* may have a large impact on the efficiency, and specific heuristics can be developed.

2.4 Conclusion

This chapter introduces the computer science background of this thesis, and presents the main techniques that are used in the subsequent chapters. Constraint Logic Programming is addressed in details, specifically regarding a constraint satisfaction problem modeling and resolution, including the three steps: variables and their domain definition, constraints elicitation and labeling. The challenges regarding the modeling and the important notions of performance and efficiency are discussed along this chapter as well.

The main programming language that is used in the thesis, Prolog, is presented and a simple example of logic program in Prolog is given. A complete resolution of the famous N-queen problem using the constraints technique is detailed for the particular case $N = 4$.

These notions are extensively used in the next chapters, where the key notions addressed already are *merged*: the gene regulatory networks and the constraint logic programming.

Chapter 3

Inferring GRN using Constraint Logic Programming

3.1 Introduction

A GRN modeling technique based on the logical method introduced in Section 1.3.5 (p11), using declarative approach and constraints, is introduced, with quite positive outcomes, by several authors (see [8, 15, 17, 50, 51]). In those work, the biological knowledge of the network and its dynamic, are formulated as a set of constraints.

The constraints paradigm seems very promising as, often, the detailed information on the cellular components and their interactions are either not available, or largely uncertain, and depend on many parameters and factors.

The formal declarative description of the biological knowledge as a set of constraints gives an interesting framework that is compatible with different matters:

- Simulation when all parameters are known,
- Reverse engineering to infer parameters,
- *A combination of both*, when some but not all parameters are known, and some data are foreboded or measured.

An alternative use of the framework can also be to simulate the network, changing one or more parameters, in order to assess the impacts of the modification.

Yet another interest for this paradigm is the capacity to deliver multiple outputs: while it is not always the case for traditional methods, constraint-based solving can output different solutions if the model is not over-constrained. There can be more than a unique model that satisfy all the known biological properties entered in the *constraint solver*. This can actually give extra information to the user, and help refining the assumptions when working on GRN's.

This chapter covers the adaptation of a biological model, based on the logical model from Thomas [62], to the Constraint Logic Programming. It enlightens in particular the variables at stake, the different constraints that can be modeled, several modeling possibilities, as well as labeling techniques.

3.2 Theoretical background - generalized logical method

Richard, Comet and Bernot, in [50], re-introduce the reader with the notions already presented in Section 1.3.5 (p11) regarding Thomas' method and logical approach. The topology of the network – one of its main properties to infer – can be described thanks to an *interaction graph* -

a notion that is used in later work as [8]. An interaction graph is an oriented graph where nodes represent genes and arrows represent interactions between genes. The arrow in labeled with the sign of the influence, indicating if the source is activator or inhibitor. A simple example is shown in Figure 3.1 (p34), representing a system where two genes are interacting. According to the formalism commonly accepted, and faithful to Thomas'initial approach, it states that gene a is an activator of gene b provided a has a concentration level higher than a first *threshold* ($+, 1$ on the arrow $a \to b$). Similarly, gene b is an inhibitor of a if its concentration level is higher than the first threshold ($-, 1$ on the arrow $b \to a$), and self activator if its concentration level goes over a second threshold value ($+, 2$ on the arrow $b \to b$).

An important precision, specifically for the reader not familiar with the domain, is that the notions of gene expression level and gene product concentration level are intimately tied together. From [50], *if* a *is an activator of* b*, then an increase of the concentration of the protein* A *encoded by gene* a *induces, generally following a sigmoidal curve, an increasing of the rate of synthesis of the protein* B *encoded by* b.

In this work, considering a gene and its products (proteins) are tied, expression level and concentration level will be employed indifferently, provided the assumptions that the expression level would refer to the gene itself, while concentration level would refer to the gene product.

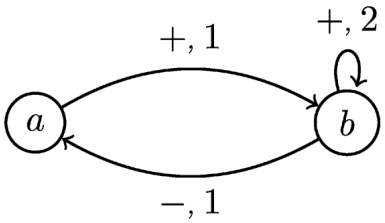

FIGURE 3.1: An interaction graph

The notion of threshold is crucial, as it states when an interaction becomes effective. The reality of the thresholds, which is not discussed in this t, leads to a discretization of the concentrations, the very basis of the logical method.

Each gene can be associated with a finite interval of integers, which corresponds to the different levels of concentration of the (product of the) gene. Simplistically, one could only differentiate the levels with boolean values, denoting the fact that the gene is either present (1), or absent (0).

3.2.1 Delay and Asynchronism

The idea of delay refers to the time required for an interaction to produce its effect, or the time for the concentration level of one product to be affected by its activators or inhibitors.

A direct consequence is an asynchronism, which is linked to the fact that the evolution of the expression level involves biological phenomena that are not instantaneous, and a priori with delays mutually distinct. It is very unlikely for two concentration levels to change exactly at the same time, and therefore, the discretization of time can lead to only a single gene expression change per time step considered. In the synchronous modeling, one could argue that still, modeling the delays - that are often unknown - represents a biological interest.

The particularity of asynchronism with respect to synchronism is widely discussed, with different views, in the literature, such as in [3, 14, 73].

It does not belong to the computer scientist to choose for the biologist this parameter - or at least it *shouldn't*, and both views need to be considered.

3.2.2 Sigmoid function and Step function

It is generally admitted that the relation between the concentration of a regulator and its effect – the increase of the rate of gene production – follows a sigmoid curve. Directly expressed in several logical modeling formalisms, such as Thomas' in [62, 64, 63], this sigmoid is approximated in logical modeling by a step function. The step function introduces a simple ± 1 depending on the concentration of the regulator which respect to the threshold. This way of modeling has been widely reused since, and will be used in the present work.

3.3 Model Variables

Gene *Level*. From Thomas'modeling, well explained in [50], each gene is associated with an integer variable which domain represents the discrete different levels of concentration of gene products: $X_a = \{0, ..., l_a\}$. In the following, and in Chapter 5 (p57) in particular, this variable will be referred to as the `Niveau` of the gene. To be able to address this concentration level over time, a specific `Niveau` is associated to a time step. For instance, $Niveau_0$ designates the concentration level at the beginning of the experiment, $Niveau_1$ at the first time step, etc. The steps being discrete, they do not represent a real timing in seconds. This is not an issue as the logical based models are qualitative.

The notion of time has to be seen in perspective with the delay and asynchronism notions as discussed here above.

Threshold. As explained in Section 1.3.5 (p11) and reminded above, an interaction between a gene a and a gene b at the step N of the experiment is only effective if the concentration level of the product of a (or the expression level of a) is above a threshold, often denoted in the literature θ_{ab}. Considering the discretization of the concentration level, the threshold is also discrete, with a domain encompassed in $[1, ..., l_a]$ where l_a defines the maximum concentration level of a. The lower bound is 1 as a bound of 0 would imply an interaction between a and b when a is absent - which is a nonsense in the real world experiment.

This threshold is at the very essence of the logical models, and the base of the main constraint to be modeled.

Delay. Considering the synchronous versus asynchronous discussion above, this work considers two ways to understand the modeling of the delays. The delay itself, as explained in [3], is supposed to be non-null, a priori unknown, parameter.

Asynchronous representation: the delays, as expressed in [50], are a priori mutually distinct, and lead to the asynchronism: at each step, only one gene sees its expression level evolving.

Synchronous representation: intuitively, it simply represents the time the interaction takes to produce its effect. For an $a \rightarrow b$ interaction, it represents the number of steps after which b will effectively perceive the influence of a. This modeling therefore allows a gene to have a delayed effect on another, while still, all the concentration levels are computed for all steps, hence the synchronous consideration. This is detailed in Section 3.4.1.1 (p36). In the general case where no delay is desired, a 0 value can be given for all interactions.

Effect. It denotes the increase or decrease of the concentration level induced by an individual interaction. It is restricted to a symmetrical domain with respect to 0, as the source gene can be inhibitor or activator, generally speaking. For instance, the `Effect` is within the $[-3, 3]$ domain. The literature often seems to limit the upper bound to 1, as in [6, 8, 50, 62], while other

references, such as Ropers in [51], allow specifying a different value, as an idea of integrating specificities of the linear differential equations. Once more, the tool built in the context of this work is not to select specifically one or another, and the way the effect of an interaction between genes is considered should comply with both ideas: the upper bound can therefore be arbitrarily set.

3.4 Model constraints

This section presents the high level constraints that will be implemented. The details are left for the Chapter 5 (p57).

3.4.1 Gene Expression Levels constraint

The very essence of the inferring or simulation work relies in the interconnection between the variables cited in Section 3.3 (p35).

3.4.1.1 Mathematical relations

The logical model generalized, as discussed in Sections 1.3.5 (p11) and 3.2 (p33), can be summed up in order to define the constraints to apply on the variables. In particular, the concentration level at a given time can be described as:

$$\mathtt{X}_j(t) = \mathtt{X}_j(\mathtt{t}-1) + \mathtt{Contribution}_j(t) \tag{3.1}$$

$$\mathtt{Contribution}_j(t) = \sum_{i=0}^{n} s(\mathtt{X}_i(t - \mathtt{Delay}_{ij} - 1), \mathtt{Threshold}_{ij}) \cdot \mathtt{Effect}_{ij} \tag{3.2}$$

where `X` denotes the concentration level, `Delay`, `Threshold` and `Effect` are linked to a specific interaction between the genes i and j, and $s(\mathtt{X}(t), \theta)$ is the boolean function which value is set as:

$$s(\mathtt{X}(t), \theta) = 1 \text{ if } \mathtt{X}(t) \geq \theta \tag{3.3}$$

$$s(\mathtt{X}(t), \theta) = 0 \text{ if } \mathtt{X}(t) < \theta \tag{3.4}$$

As clearly shown, there is a form of logical `OR` intrinsic to the modeling : either the concentration level is greater than or equal to the threshold, leading to the effectiveness of the interaction, or the concentration level of the source gene is smaller than the threshold - which may be unknown, in the most general case - leading to a null contribution of that particular gene.

These mathematical expressions will be used, and their implementation detailed in Section 5.4 (p60).

At this point, a disclaimer *needs to be addressed. Equations 3.1 (p36) to 3.4 (p36) are set as the output of the unification of several biological studies listed in references: as stated along the first chapters, not all methods agree, and differences between authors can be from cosmetic to drastic. As for example, the synchronous versus asynchronous modeling, that both have sponsors across biologists community - also due to the ultimate use of the models. Once more, although having the most accurate possible representation of the up-to-date knowledge of the GRN inference is important, it does not prevent this work for showing the interest of the Constraint Logic Programming techniques.*

Asynchronous case. Equations 3.1 (p36) to 3.4 (p36) are used as is for the synchronous case. While considering the asynchronous modeling, things get quite different, as only one interaction is to be considered at a time. The *time* precisely does not really follow a real time reference, but rather indicates the sequence in which the interactions are applied. The asynchronous case may seem easier, as only one interaction is defined at a time step, but is trickier when considering a concrete interpretation.
The literature, such as Thomas in [63, 64] suggests and explains that delays are most likely mutually distinct and only one interaction between genes need is to be considered at a time. It is also generally assumed that the delays are unknown : they appear as perfect candidates for the CLP paradigm. Nevertheless, the challenge consists in selecting properly which interaction to choose at a given time. This topic raises several questions, not answered at the time of writing these lines, such as how to model the fact that in general, there is no reason why all the interactions should be applied one after the other, in a *unique* pathway. Because of this delay, the time that an interaction takes, some fast interactions could be applied several times before another one, slower, becomes effective the first time. As an illustration, let's imagine three interactions $a \to b$, $a \to c$, $c \to b$, associated respectively with D_{ab}, D_{ac}, D_{cb}, the distinct delays. Let's consider that $D_{ab} < D_{ac} < D_{cb}$.
In the most general case, depending on the *real* delays and most likely also on many other parameters, all the following pathways could be acceptable:

- $a \to b$ then $a \to c$ then $c \to b$
- $a \to b$ then $a \to b$ then $a \to c$ then $c \to b$
- $a \to b$ then $a \to c$ then $a \to b$ then $c \to b$
- $a \to b$ then $a \to c$ then $a \to b$ then $a \to b$ then $c \to b$
- ...

In other words, there is no reason why the first pathway would happen, as the first interaction could apply its effect several times before the second one is realized.

As a further question, what happens after $c \to b$ is effective : does the pathway start again from the beginning of the sequence, or does a complete other pathway interact ?

That is, it is unclear[1] whether all the interactions of a GRN are to be taken in a single order, or if the delays may change, or if the pathways change, etc. Following the disclaimer presented above, these considerations specifically directed to the modeling of the GRN in the logical world will not be investigated any further in this work. To go on with asynchronous models, however, the following assumptions will be taken:

H1 the delays, in the asynchronous case, are distinct,

H2 the delays are unknown, although the user can suggest a (partial) order,

H3 the interactions are applied following a specific sequence, defined by the delay. Using the same example as above, over a period of 6 steps of time, the sequence of interactions application would be $a \to b$, $a \to c$, $c \to b$, $a \to b$, $a \to c$, $c \to b$.

H4 the gene expression level to be compared to the threshold will be the one from the previous step.

[1] From the author perspective, to say the least, most likely due to a lack of biological knowledge, although the biological sciences may still have discoveries to make.

Intuitively, as only one interaction is active at a time, it won't be surprising to observe phenomena taking much more time steps than in the synchronous case.

Another modeling possibility, left for the perspectives in Chapter 8 (p105), for the asynchronous case could be the following: For each step N, for each gene g of the system, are considered all the potential interactions i that could start, as the expression level of g is higher than the threshold θ_i. All those potential interactions are then sorted according to the `Delay` variable. The next i to occur is the one with the smallest `Delay` value.

3.4.2 Sparsity constraint

This extra constraint, introduced in [15], comes from the idea stated in Section 1.3.2 (p9) that usually, only a (small) subset of genes are regulators, and the overall amount of interactions is limited. At first, it is considered that all genes can have an interaction with all the others. This constraint allows specifying a maximum number of interactions.

For the sake of clarity, this way of modeling could be improved : the sparsity constraint as foreseen here limits the number of interactions, and not the number of regulators. This could constitute an improvement, if judged necessary by domain-related users.

Adding this constraint, which should be optional in the most general case, may not help finding a solution, but could definitely limit the total amount of acceptable solutions.

3.5 Labeling

As introduced in the Section 2.2.5 (p24), labeling is the part of the resolution of a CSP where the variables are assigned a value, based on specific variable and value ordering.

Regarding the application domain and its modeling formalism adopted, five types of variables require a labeling: the `Niveau`'s, the `Effect`'s, the `Threshold`'s, the `BorneEffect`'s[2] and the `Delay`'s.

The concrete implementation of this labeling is detailed in Section 5.4.8 (p70), but key indications are given in this section. Based on theoretical considerations, several ordering techniques for both variables and value can be foreseen.

3.5.1 Variable ordering

A first heuristic to apply is to use the broadly efficient *first-fail* technique : the variable to be selected first is the one with the smallest number of elements in the domain.

A second variable ordering technique is to label first specific type of variables, such as the `Niveau`'s and the `Effect`'s, as they are intuitively the most constrained variables, while the `Threshold`'s and `BorneEffect`'s would just adapt to the defined values. In this case, the assignment of the `Niveau`'s and `Effect`'s will always be done before the other variables. The backtracking algorithm will therefore first try out the choice points left for those last variables, if no other specific dispositions are taken. This is discussed in Section 5.4.8 (p70).

3.5.2 Value ordering

The value ordering problematic lets appear very interesting notions in the domain of interest.

The first ordering technique, usually the default one, consists in assigning the value from the lower bound to upper bound. When all the values have equal chances of being part of a

[2]These variables simply stand for the bounds of the `Effect`'s domain. This is introduced longer in Section 5.4.3 (p63)

real-world solution, that is, when the probability of all the values of the domain to be part of the solution is the same, there is no specific reason why the order would matter regarding the relevance of the solution found, and the values ordering can simply be guided by the efficiency to find a solution. For instance, if the problem modeled is related to assigning seats around a table, provided all the constrained modeled are respected, one can assume there is no better solution than another.

On the contrary, if the modeled real-world problem is under-constrained – leading to several solutions – and if it is known that some solutions are more relevant, the value ordering is a technique to have those more relevant solutions appear first.

This is interesting, as it can be understood as an integration of a probabilistic framework over the labeling: by selecting appropriately the values assigned to variables in a way that respects the known or estimated probability of appearances, the optimal solutions - the ones that are given first by the solver – will be the most relevant ones.

The GRN modeling domain is definitely impacted by this promising behavior, as confirmed by the probabilistic modeling considered in Section 1.3.5.3 (p13) or described by Georgoulas and Sanguinetti in [21]. Of course, ordering the values so that the most frequent ones are tried out first does not intend to fully integrate a complete probabilistic model.

In particular, the ordering of the values to be assigned to the `Effect`'s variables can appropriately selected, as it is foreseen that solely a small number of genes participate to the regulation.

The implementation of the different ordering techniques is detailed in Section 5.4.8 (p70).

Finally, although the labeling may suggest prospective more relevant solutions first, it does not not prevent - in the general case at least - to find all the others.

3.6 Modeling GRN with constraints in the literature

The main overall idea, specifically described in [8], is to start from Thomas' logical model of the network (see Section 1.3.5 (p11)), adding several more complicated features such as delays or asynchronism, and to determine all the possible *states* for the system. Then, applying constraints, paths between the starting and steady states are computed, in such a way it gives how the system necessarily evolved under the constraints. Recently, Fromentin, in [17], or Fioretto [15] suggested different constraints modeling, as using community networks to integrate predictions from individual methods (presented in Chapter 1 (p3)) in a "meta predictor", in order to compose the advantages of different methods and soften individual limitations. This is out of the scope of this thesis.

As also notified in [8], they suggest to relax some constraints if no path are possible. Although this technique shows good results to reason on the dynamics of networks, our work will not focus on the stable states of the system, but rather directly on the gene expression and on the gene products concentration. That is, rather than finding paths and successors to states to go from starting genes concentration levels to steady states, or to find all possible states of a system, this work will focus on the tool described at the end of Chapter 1 (p3), using the introduced constraints framework to describe the biological knowledge.

3.7 Conclusion

This chapter describes how to build a constraint logic program starting from a biological model. It goes into detail in the theoretical background required to understand the context, then it addresses the way to convert it in a CSP using the three steps: variables elicitation, constraints elicitation, and labeling. The reader should be familiar with the notions described in this

chapter as they really are the concrete base upon which the whole implementation is built. In particular, the variables and their semantics is important. The CSP implementation detailed in Chapter 5 (p57), detailed in the subsequent chapters, abundantly refers to these notions.

Chapter 4

Tool Definition

4.1 Introduction

Considering the needs for the *biologist*[1] expressed at the end of Chapter 1 (p3), a tool developed in the context of the present work.

This chapter first introduces the reader with the major requirements for this tool, at different levels. For that purpose, Lucy, a typical biological-user, is introduced as the reference *persona* (see Section 4.2 (p41)). Secondly, the architecture of the suggested tool is discussed, and chosen. The technologies to be used for the implementation are selected, based on the technical needs, the availability, the timing constraints and personal ability.

The user inputs required, based on Chapters 1 (p3) and 3 (p33), are listed and detailed.

As a result, at the end of this chapter, the reader will have an overview of the global structure of the tool built, its use, its architecture, and its technology.

4.2 Requirements Overview

This section provides the main requirements of the tool. The goal is not here to apply a specific software development process, such as the "V" development cycle or following Agile method – this would have little interest in the context of the work. It rather gives an informal description of what is expected to be provided. Subsequent sections detail how the tool fulfills those requirements.

The very succinct *persona* (as defined in [45]) of the expected biologist user is Lucy: a biology aware person, used to work with a personal computer for administrative or office tasks, or able to use experts software in its field of activity. Presumably, Lucy has no programming experience, and is definitely not aware of any constraint logic programming paradigm as presented in Chapter 2 (p19). The tool is dedicated to help Lucy in her work related to GRN's. The tool should:

- be able to infer gene interactions based on user inputs,
- be able to simulate and generate data based on interaction rules defined by the user,
- be able to fill the gaps in rules and data: based on a subset of rules and a subset of data, reconstruct the complete set of rules, and set of data.

The three requirements listed above can be refined and expressed directly as a user story [71]:

- As a user, I want to provide the entities, part of the system I want to study with the tool.

[1] generic word designating a student, an expert – anyone interested in the field of gene regulatory networks

- As a user, I want to provide data so that the GRN can be inferred.
- As a user, I want to provide the interaction rules between genes so that data expressing the concentration levels of the gene products can be generated.
- As a user, I want to be able to manually modify the interaction rules suggested by the tool.
- As a user, I want to be able to see the results of the inference or data generation.

From a technical perspective, as it also consists in the main topic of this work, the tool should integrate the constraint logic programming paradigm (see Chapter 2 (p19)).

4.3 Architecture

4.3.1 Standalone or Web based

To build this tool, several architecture types are possible. One could think first to develop a tool based on a standalone executable, that Lucy can install and run on her own machine. Alternatively, the tool can rely on the web service idea : Lucy would only need a web browser to access to the tool.

The choice of one or another solution is mostly linked to non-functional requirements, as both architecture could handle the functionalities listed in Section 4.2 (p41).

4.3.1.1 Standalone application

As defined in [2], a standalone application can be defined as any software program that does not require anything else in order to run, provided the environment is compatible (for instance: Java is installed on the machine). Essentially, it is a software that can stand on its own without help from the Internet or another computer process. From other sources, the definition may slightly differ, but the essence remains.

The main advantage of the standalone application, from the user perspective, is the independence of use with respect to an Internet connection. Installed on a limited environment – Lucy's computer – the standalone application does not require external access to process data or provide its services. This is however not true if the application requires a connection to an external database or other external service to properly run.

A second advantage is the security – or at least the control – offered: it is easier to limit the threads on an individual machine.

From a programming perspective, a standalone application necessarily means to handle the different architectures it will need to run on: different OS, different HW material (laptops, desktops, mobile devices,...). This results in a considerable overhead in terms of development but also test, as all the configurations should be tested prior to official release. Of course, appropriate technological choice can reduce the overhead, such as relying on the JVM[2] to run the application, making it *portable*. Yet, the different configurations should be tested.

For the user, despite not needing an Internet connection, the ease of use of a standalone application may not be optimal. Among some common issues, one can cite:

- problems related to specific machines (hardware/software compatibility),
- permissions and rights issues – specifically on machines managed inside a larger organization, where the individual may not have the rights to install third party software,

[2] see [60]

- resources issues, for lickerish applications (ex: CAD Software often need powerful computer, with many GB of RAM to guarantee a correct user experience).

Regarding the maintenance, all the individuals that have their version of the software installed will need to *regularly* update the applications. As commonly admitted, this cumbersome aspect often leads to non-updated software versions still running. In regards of the security aspect mentioned above, this can on the contrary constitute a risk, as patches and updates also deal with security reinforcement. Leaving software running that are not up-to-date in terms of security updates can cause breaches.

Finally, imposing as many downloads and installations as users may limit the spread of the software developed.

4.3.1.2 Web application

At the very opposite of standalone application, a Web application is a computer program that uses web browsers and web technology to perform tasks over the Internet, as defined in [80].

Web applications are installed in a unique (or considered unique) place, a server, and the users – usually called the clients, can access it traditionally through their classical Web browser, using their Internet connection.

This simple definition may be declined in different ways or detailed in terms of web technology used for instance – while the essence still remains the access to a service using web technologies.

If several years or decades ago, having an Internet connection and data bandwidth allowing large amount of data being exchanged between client and server was not always easy, it's rarely an issue nowadays – for most of the users at least, specifically in the context of Lucy's work.

Web applications present the main advantage of the ease of use for the user : Lucy does not need to care about her configuration on the *device* she uses, as only a web browser is required[3].

Regarding the installation and updates, web applications need to be installed only once, on the server side. From a *maintainability* perspective, this is a major advantage.

Security-wise, web applications are much more at risk than standalone applications as they are intrinsically more exposed, being open to the web. Several well-known tactics and tools however help the programmers in deploying in security the application, but yet, it remains a challenge to ensure the continuity of service and data security while forbidding access to unauthorized clients. As always, the needs of security (all aspect wise) of a particular application depend on its security risk analysis.

Web service. A distinction needs to be made between Web service and Web application, although depending on the source, the information may vary and both may be used in the context of this work. A Web service is the software on the server-side that exchanges information – sometimes processed – via a standard web protocol, typically HTTP. The interface offered by the Web service is called API. The Web application is considered the software that uses this API, provided by the Web service, typically to show the information to the end user. Usually, the Web service itself is not destined to the end-users, but rather to another software such as a Web application.

4.3.1.3 Conclusion

The previous paragraphs gave a succinct indication on when using standalone application or when using web application/web service, mostly depending on the non-functional quality attributes such as maintainability, security, ease of use, resources consumption, etc.

[3]One may however argue that yet, Flash, JavaScript or other packages would need to be up and running on the client side. This does not really go against the ease of use when compared to standalone application

Based on the facts that the intent is to have:

- a tool widely spread among Lucy's,
- an effortless integration in Lucy's work environment,
- an easy-to-test tool without requiring time to install,

and considering that the security and connectivity are not a limitation, the choice of a web application seems the most appropriate.

Figure 4.1 (p44) depicts schematically the Web application principle as considered here. Users located behind several kind of workstations, from a desktop to a tablet, provided they have access to a Web browser, can access through the Internet to an application server, where are actually installed the services the users want to access. Behind this schematic view of the application server is in reality a much more complicated architecture, integrating all the specific functional and non-functional needs for the application[4].

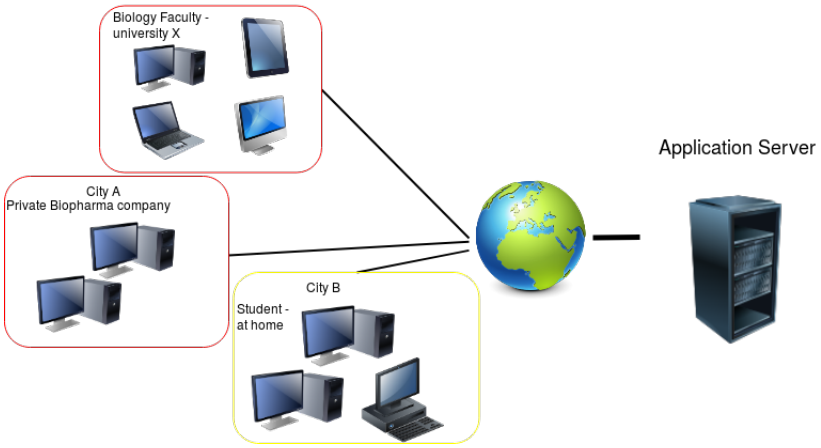

FIGURE 4.1: Web Application - schematic view

This overview is purposely limited in terms of components.

This work aims at providing a prototype of the tool rather than a production-grade software, and the goal is to establish the base, making sure the choices are motivated, without digging into the details of the application implementation.

The distinction between Web service and Web application is important: the focus is primary put on the server side, the Web service, rather than on the client side, the application.

As a consequence, the tool built as part of this work will not address the following topics:

- sessions or security-related,
- graphical usability,
- data persistence (connections to databases),
- quality attributes related to production mostly, as the scalability, the reliability, ...

Still, those topics are clearly important for real and in production web application. Chapter 8 (p105) will discuss several suggestions to convert the prototype to a *real* product.

[4]For instance, load balancers hardware or software, firewall

4.3.2 Style

The Web application as used for the tool prototype is based on a client-server architectural style (see [12]). The client, used by Lucy, invokes a service of the server component.

The tool is divided in two parts: the server part, called back-end, and the client part, called front-end. The server part also corresponds to the Web Service definition.

The Web Service developed reuses the principles, without calling itself fully compliant, of a REST API. The REST style was initiated by Fielding, see [13]. The objective is not to fully describe and discuss the style, but rather to informally summarize the major principles that make REST a widely used and appreciated style.

REST stands for Representational State Transfer, and is governed by several principles:

- **Client-Server** As already discussed implicitly, the client shall connect to the server, exchange information and receive a response based on its request. This is typically a Web Service Architecture style. It reinforces the separation of concerns principle, with help with the portability of the client (as the server remains identical on another platform, for instance), the modifiability of the parties, as their internal modification has no impact on the others. Only the interface, or API in the field of Web services, needs to be modified.

- **Stateless**: REST is based on the Client-Server stateless idea that the request of the client is self-contained, and the result should not depend on the server context. As no information is kept on the server side, this implies an increase of the data exchanged during a request. Besides this, stateless style has proven itself to be easily scalable (it can be easier to integrate load balancing techniques, as the requests are self-contained), reliable (independent of the server state), and easier to debug since there is no context to take into account.

- **Layered System**: REST allows n-tiers style, while traditional client-server style are generally 2-tiers. The client should know to what server it is connected. Layered – and multi-layered – architecture can allow resources sharing and scalability improvement.

- **Cacheable**: optional, the response to a request can be labeled, implicitly or explicitly, as cacheable or not. If cacheable, it means that the client has the right to reuse later the same data as response for the same (or equivalent) request. This cache can have a timing validity and be updated if required.

- **Uniform Interface**: the messages shall be self-descriptive and the request self-contained, the resources shall be identified, and their manipulation done through representations.

These principles will be partially followed. The tool is built based on a stateless client-server style. It uses HTTP protocol to communicate with clients, and the communication are based essentially on the HTTP POST command. The data format of the resources exchanged is discussed in Section 4.3.3 (p45).

The multi-layered design is not needed *yet* in the context of the prototype. Besides, the first version of the tool will not support the cacheable constraint, while improvement can be made without fundamental changes.

4.3.3 Data Exchange Format

As the client side and the server side will exchange information, the interface needs to be well specified including the format in which the data will be sent/received.

The data exchanged are largely detailed in Section 4.5 (p51). They include the complete knowledge for the server to perform the GRN inference as expected, hence the stateless property.

As indicated in specialized Web services resources such as [78], the two most common data exchange formats are JSON and XML (or XML-derived formats such as WSDL)

XML. XML stands for Extensible Markup Language. It is used to define documents with a standard format, that can be read by all XML-compatible application. The tags are not predefined (like HTML tags are). Originally, XML was designed to carry data, with the focus on what data are, as specified in [84]. An example from [34] is given here under.

```
<computer>
  <name>Gaming PC</name>
  <components>
    <cpu>Intel i7 3.4GHz</cpu>
    <ram>16GB</ram>
    <storage>2TB HDD</storage>
  </components>
</computer>
```

JSON. JSON stands for "JavaScript Object Notation". Specifically designed to transmit structured data, it has imposed itself for data transfer in web application and web servers, as confirmed in [34]. Its main strength is the compactness of the representation, specifically with respect to XML. The encoding of the previous example in JSON becomes:

```
{
  "computer": {
    "name": "Gaming PC",
    "components": {
      "cpu": "Intel i7 3.4GHz", "ram": "16GB", "storage": "2TB HDD"
    }
  }
}
```

Numerous online sources describe the pros and cons of JSON and XML to exchange the data (for instance, [85]). In the present work, considering the lightweight idea of JSON, its increasing popularity for web development, the easiness to set up at first the test bench, the accessibility of the documentation, and personal habits, JSON has been selected as the data exchange format between the server and the client.

Several other formats typically XML based, were developed a few years ago in specific context – some of them in the field of Biology and Biochemistry. Such formats are for instance SBML[5], CellML[6]. Considering the context, it has not been judged relevant to use such a format as

- they are not common outside of the scope of their applicative domain. For instance, BioPAX[7] is specialized to describe biochemical networks, and doing so can be used in the context of GRN. But BioPAX is also unknown in other network domains.
- even though they are XML based, there are not many dedicated parsing tools,
- there exist a multitude of formats: the goal of standardization using one of the others is not achieved yet,

[5] **http://sbml.org/Main_Page**.
[6] **http://www.cellml.org**.
[7] **http://www.biopax.org/**.

- it may be more efficient to develop the service based on a real standard interface and add a layer of transcription (compatible with the REST style) that would convert specific format into JSON, then back to specific format, rather than implementing different interfaces for all the potential biological-related data exchange formats.

4.4 Technology

The suggested solution gives a lot of freedom for the implementation. The back-end (server side) and the front-end (client side) can be handled in many ways: it reinforces the separation of concerns principle. Chapters 5 (p57) and 6 (p91) detail the implementations.

Many tools/technologies are – at the time of writing those lines – available for a programmer to implement reusing code rather than starting from scratch. On the front-end side, this code reuses can go equally from a Javascript library that ease the visualization of networks, such as **vis.js**, to a complete framework such and **Angular** or **React**, integrating natively many functionalities. On the back-end or server part, code reuse techniques also involve frameworks such as **Spring Boot**, a kind of simpler and more integrated **Spring**, or **Play**, quite recent but with extensive references already. Many others yet exist.

The following sections aim at discussing the possibilities and defining the selected technologies for both part of the tool to build, front-end and back-end.

4.4.1 Front-end technology

The Front-End of a website is the part that users interact with. It consists in the software code that is used by the client to request the service, enter its inputs, view the outputs, ... In the context of this work, the tool to be built requires a front-end in order to call the Web service developed. Indeed, if the back-end can essentially be called using HTTP request, tools like cURL or Postman can theoretically be used. However, it appears obvious that it does not consist in a viable and easy-to-use solution, even for a prototype, and a dedicated front-end should be provided. The details of the implementation will be covered in Chapter 6 (p91).

Nowadays, all the content that can be viewed when navigating on Internet, using web browsers, is a mix of three languages:

HTML: it defines how *every* website is organized, and is the backbone of the front-end technology. Thanks to HTML, the desired content can be added on the web page. It is a markup language (same idea as XML, introduced above) with dedicated tag. The current version is HTML5, recently updated. The complete specification can be found on the W3 organization website [76].

CSS: it defines the style of the HTML document, its layout – how the elements are displayed on the screen. This really is related to the colors, the size, the shapes, ... The interested reader can find much information on the W3 organizationwebsite [75].

JavaScript: it is used to program the behavior of the web pages, to add dynamic content or react to user actions. Extensive documentation is provided on the W3 organization website, [77].

Evolution of the cited languages, and JavaScript in particular, makes front-end quite powerful, and a lot of computation can now be made on the client side, rather than on the server side. It helps having a dynamic interfaces – as single page application – , as the server does not need to recompute and send a whole new page but rather the client refreshes the current page without reloading static content, for instance. An example is the Ajax technique, with which

the page is dynamically refreshed by the front-end while data are downloaded from the server in the background, which allows single page application. More information is available online, in [74].

Although the languages selection seems standardized, building front-ends has been considerably improved by several frameworks and libraries that help the programmer in finding solutions on common problems, or provide bunch of code dedicated to specific function. For instance, a framework can have a built-in strategy to ensure good-looking content no matter the device used (a laptop, a desktop, a smartphone, a tablet,...), or it can natively include a common login form, etc. As pointed out in [65], the main purpose of those frameworks is to serve as skeletons, specifically for single page applications. They let programmers focus on the interface elements, using JavaScript and HTML, without worrying about the overall code structure or maintenance, which is encapsulated by the frameworks. Using a *well-chosen* framework can help:

- the maintenance and structure of the code, following recognized guidelines,
- using piece of validated codes such as common login forms,
- using security features, built-in,
- focus on the interface elements rather than on the structure details[8], without needing any action from the programmer,
- build single page application.

Single page application has become one of the most popular ways to develop web applications. As already introduced above[9], it can significantly improve the response time, as a new page is not reloaded at each request: only the modified element is fetched.

It is simply impossible to list all the frameworks that are *currently*[10] used, and specialized websites are constantly running benchmarks and publish detailed comparisons. The most popular frameworks at the time of writing these lines seem to be **React**, **Angular**, **Ember** and **Vue**, while still many others exist. The interested reader will find a comparison between the four main players in [65]. A complement is also given in [66]. In [41], an informal while relevant map is given. This map sets some frameworks in perspective, as the choice can be crucial.

At the bottom of it, even the essence of the choice need to be assessed, as using a framework represents always an overhead: it shall be worth it. It may not be adequate for all situations, specifically for prototyping, if the framework is initially unknown. In conclusion, the decision of using a framework or plain HTML/CSS/JavaScript and the selection of this framework are a matter of the use cases, the preferences and the quality attributes that are looked for. If production-grade front-end may hugely take benefits of a framework, for which the return on time investment will stand in no time, it may not be the case for simple interfaces.

Back to the context of this work, hence, the goal is to provide Lucy a simple interface, fulfilling the very basic requirements as in Section 4.2 (p41). No need for fancy interfaces, extremely dynamic behavior, or integration of complex patterns. However, Ajax techniques can be used to benefit from single page application idea, and several libraries – rather than frameworks – will be used to help and ease the implementation. This will be detailed in Chapter 6 (p91). At this point, it is relevant to point out that the separation of concerns principle followed in the architecture style choice, with separated front-end and back-end, helps

[8] For instance, the structure of the codes is naturally split following the MVC pattern[42]

[9] When discussing the Ajax technique

[10] It indeed changes rapidly

developing this simple interface without limiting evolution as another front-end could interface with the exact same back-end, providing the JSON data exchange is respected.

4.4.2 Back-end technology

The back-end consists on the server side of the application, the part that is not directly in contact with the user, or client (front-end, see Section 4.4.1 (p47)). It consists mainly on the server software and the data persistence. The back-end, in the previously selected architecture stateless, client-server style, is the code that will receive and understand the requests, process the data according to the request, then set up a response that will be sent back to the requester.

An interesting schematic view of the components involved is given in Figure 4.2 (p49). The source of the image is the Upwork website, [1].

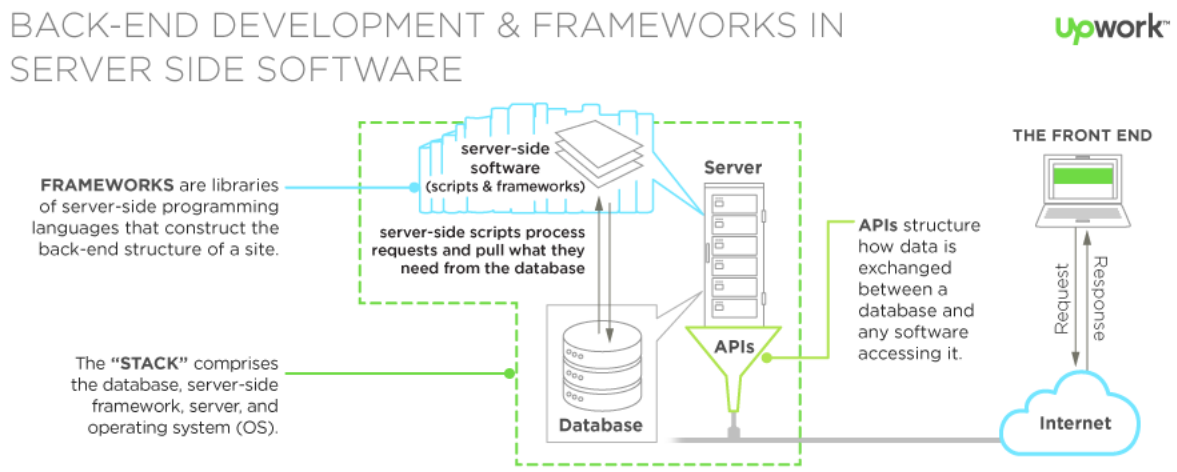

FIGURE 4.2: Schematic view of a back-end development architecture

Back-end development includes, as shown in Figure 4.2 (p49) the API part, the code that *does something with the request and resources*, business related and core of the intelligence of the server, and then usually a persistence layer involving one or more databases. Similarly to the front-end development technologies, the back-end development can benefit from multiple frameworks, all of them having their own particularities and pros/cons.

Specialized websites often classify the server development frameworks, and a major distinction is usually the programming language used. By contrast with the front-end languages, multiple programming languages can be used for back-end development. A vast choice is offered: some languages are Object-Oriented, others are compiled rather than interpreted,... Choosing a language implies several consequences regarding the ease of implementation, the non-functional attributes, the execution time, the control, etc. The following list is based on current popularity, according to [1] and [27]. Examples of corresponding framework are given as well. The following list is not exhaustive and many others exist.

- Java (Spring Boot, Play, ...)
- Python (Django, ...)
- Ruby (Ruby on Rails,...)
- JavaScript (Node.js, ...)

- PHP (Symfony, ...)
- Scala (Play, ...)

As for the front-end, the choice of framework and programming language has an impact on the back-end development.

As Java is commonly used at the University of Namur, two Java-related frameworks will be succinctly reviewed.

It is important to keep in mind, reading these sections, that in the scope of this work, as described in Section 4.3 (p42), the tool developed will not include a data persistence layer, and will limit the requests handling part to the minimum required. The core of the back-end is the *business related* constraints logic program.

4.4.2.1 Spring Boot

Spring Boot, see [56] is a Java framework for building Spring-based applications, and web application in particular. Spring Boot is *"designed to get you up and running as quickly as possible, with minimal upfront configuration of Spring"*.

One of the main force of Spring Boot is the community of developer around it and the abundance of documentation, how-to's, but also "native" functionalities offered by the framework. In particular:

- an initialiser, "Initializr", that allows to select what features to integrate to the working project, in order to minimize imports later on,
- tools and classes to *easily* implement REST API's, Streaming, WebSocket, ...
- integration of a dedicated Security environment, [44], that helps in the production-grade security requirements,
- wide support of many database systems (SQL and NoSQL)
- ...

Spring Boot is broadly used worldwide, and comes with many features built-in, which, once more, encourage the code reuse, hence improve its quality.

Although impossible to fully describe in a few lines, Spring Boot consists in a complete environment that can support most of the needs of application or Web applications.

4.4.2.2 Play

With the same fast bootstrapping as Spring Boot, the Play framework intends to help building a Web application in Java or Scala easily, with high velocity. The documentation is available in [46]. In particular, it encloses natively many relevant features to help developing RESTful API such as JSON support, stateless behavior by default, asynchronous request, etc.

Furthermore, it includes rapid application development, where the code running on the server simply needs a browser refresh to be applied – and no server restart. This is a time saver during development phase (which never really ends).

The documentation and example projects, while being not as abundant as for Spring Boot, is accessible and dedicated how-to's are provided to start-up quickly the application.

As Spring Boot, Play comes with a multitude of ready-to-use functionalities including security aspects, data persistence tools, etc.

4.4.2.3 Prolog

Considering the use of Prolog for the part of the system related to Constrain Logic Programming, as indicated in Chapter 2 (p19), it is interesting to note that the chosen implementation, **SWIPL**, comes with a specific WEB application environment, detailed in [70]. Among other features available, it defines or helps in particular to

- receive and answer to HTTP Request,
- handle JSON inputs/outputs,
- deal with classical CORS aspects, see [9].

Many other aspects such as authentication, session handling, data persistence and connection to a SQL database, ... are also covered in the tutorial.

The main drawback, as all development done in Prolog, is the lack of IDE to ease the developer task: everything is coded using the programmer's favorite text processing software. This is beautiful as no complicated machinery or configuration is required, but it also involves some tasks or errors usually taken care of by common IDE's.

4.4.3 Technology - final words

It is imperative to remind that the main focus of this work resides in building a prototype tool using Constraint Logic Programming to reason over Gene Regulatory Network. It is not to fully implement a production-grade web application. Corollary, most of the usual quality attributes related to WEB development are not fully covered, neither the tactics to respect those non functional needs.

Similarly, in order to stay focus on the main mission and not spreading the resources available, it has been decided to drastically limit the front-end part to what's required to imagine a possible product – and to spare the back-end or server part from important but cumbersome work in this context.

These considerations have guided the choice of technology use:

Back-end: The server side is completely in Prolog, using dedicated HTTP module in particular to handle the REST Style. The back-end implementation, including the CSP, is detailed in Chapter 5 (p57).

Front-end: HTML, CSS and JavaScript, using relevant libraries when useful, will be used. The front-end implementation is detailed in Chapter 6 (p91).

4.5 User Inputs

The previous sections have enlightened what an API is, and selected JSON data exchange format as the API formalism in Section 4.3 (p42).

This API consists in exchanging the information required for the server to understand and execute the request, and for the front-end to receive, process and set up appropriately the server's response to the user.

Given the context, a DSL[11] has emerged, in order to properly describe the networks. From the front-end to the back-end, the user inputs are given, while –using the same format– the response integrates the server processing.

[11] Domain Specific Language

To contextualize, a network as represented in Figure 4.3 (p52) is to be inferred using the tool. The corresponding JSON file is given here under. The network consists in three entities, two of the *node* type, and one of the *control* type. The interactions and their effect are visually represented in Figure 4.3 (p52).

Despite specific nodes and edges, several global network parameters need to be given – which explicit either global boundaries and thresholds that are applied as limits by default – as well as other solving parameters such as the labeling method to apply, or the modeling method. The complete details of the information in the JSON exchange file are given in Table 4.1 (p55).

Altogether, these JSON properties allow defining precisely the network under assessment and the task to perform.

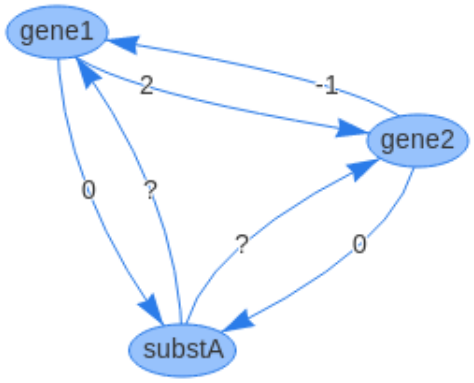

FIGURE 4.3: Network represented as a graph


```
{"network":[
{
"name":"yourName",
"borneMax":10,
"borneMin":0,
"borneEffectOnOthers":5,
"borneEffectOnSelf":0,
"globalThreshold":9,
"steps":4,
"method":"lineaire",
"sparsity":0,
"labeling":"all_ff",
"nSol":1,
"nodes":[
{"label":"gene1","type":"node"},
{"label":"gene2","type":"node"},
{"label":"substA","type":"control"}],
"edges":[
{"id":"toChange","from":"gene1","to":"gene2","threshold":4,
        "borneEffect":5,"delay":0,"effect":2},
{"id":"toChange","from":"gene2","to":"gene1","threshold":"/",
        "borneEffect":5,"delay":0,"effect":-1}],
"data":[
{"node":"substA","step":4,"niveau":2},
```

```
{"node":"substA","step":3,"niveau":4},
{"node":"substA","step":2,"niveau":4},
{"node":"substA","step":1,"niveau":2},
{"node":"substA","step":0,"niveau":0},
{"node":"gene1","step":0,"niveau":5},
{"node":"gene2","step":0,"niveau":0}]}]}
```

Parameter	Semantic and acceptable values
`name`	This is the name of the network as given by the user. It can include spaces, "_" or numbers. The core of the back-end, the CLP, does not use the name.
`borneMax`	Maximum value that any concentration level can have. Typically, for a boolean network, the borneMax is 1. If the concentration level is expressed in percentage, borneMax is 100. Any positive or null value can be given. The largest borneMax, the largest the solution space is, and the longest may take the search.
`borneMin`	Minimum value that any concentration level can have. Typically, for a boolean network, the borneMin is 0. Any positive or null value can be given. The concentration levels will be bounded by $[borneMin, borneMax]$
`borneEffectOnOthers`	Maximum effect that a gene product can induce on another gene. This upper bound is a positive or null integer.
`borneEffectOnSelf`	Maximum effect that a gene can induce on itself. This upper bound is a positive or null integer. In the first version, the self effect are unfortunately not effective.
`globalThreshold`	Maximum value that any threshold defined in the edges can take. It consists in an upper bound. The globalThreshold is a positive or null integer.
`steps`	Number of time steps during which the CSP needs to perform the inference. The largest the steps, the longest may take the search.
`method`	It denotes the modeling method to apply. In the current version, the possibilities are : `lineaire`, `tmp`, `memoisation` or `asynchronous`. Often, `lineaire` is a sensible choice. More information on these methods in Section 5.4 (p60).
`sparsity`	It denotes the maximum number of interactions between genes, as described in Section 3.4.2 (p38) and described in Section 5.4.7 (p69).
`labeling`	It denotes the labeling method to apply. In the first version, labeling can take the values `all_ff` or `optimized`. More information in Section 5.4.8 (p70).
`nSol`	It is the number of solutions that will be computed by the CSP. It consists on a maximum: if nSol is larger than the amount of all possible solutions, only the possible solutions are sent to the user. nSol is a positive integer.

`nodes`	It consists on an array of all the entities that belong to the system. The entities shall be added according to the JSON format, as shown in the example file.
`label`	It denotes the name of the entity. It is used directly by the CSP : it starts with a lowercase letter, and does not contain any space. This serves as an identifier for the entity.
`type`	It denotes the type of entity that is added. Two choices are offered: `node` and `control`. A node substance is constrained according to the modeling discussed (see Section 3.4.1.1 (p36) and Chapter 5 (p57)). A `control` substance does not have its concentration levels constrained by the regulatory network models. Its concentration is supposed arbitrarily controlled.[12]
`edges`	It consists in an array containing all the interactions description
`id`	It refers to an ID for the current interaction between entities. This parameter is not used in this version of the system, and is automatically replaced. It may be used in the future.
`from`	It denotes the source of the interaction. It shall be the label of any of the entities contained in nodes.
`to`	It denotes the target of the interaction. It shall be the label of any of the entities contained in nodes.
`threshold`	It denotes a specific threshold for the interaction. The threshold is an integer between 1 and the `GlobalThreshold`. If it is unknown, the user can input `"/"` that is a special character for the CSP.
`borneEffect`	It denotes a specific symmetric bound for the interaction. The effect of the relation will be bounded by $\pm$`borneEffect`. It is a positive or null integer, lower than `borneEffectOnOthers`. If it is unknown, the user can input `"/"` that is a special character for the CSP.
`delay`	It denotes the delay of the current interaction. It can have different meaning. More information in Chapter 3 (p33) and Sections 5.4 (p60) and 3.4 (p36). If it is unknown, the user can input `"/"` that is a special character for the CSP.
`effect`	It denotes the impact that the source gene may have on the target gene, according to the mathematical relations in Section 3.4.1.1 (p36). It is a positive or null integer. If it is unknown, the user can input `"/"` that is a special character for the CSP.
`data`	It consists in an array containing all the concentration levels data description.
`node`	It is the label of the entity for which the concentration level is given. It should match one of the label in `nodes`
`step`	It is the time step for which the concentration level is provided.
`niveau`	It is the concentration level input of the node at time `step`.

[12]Note that if a concentration level is not given as input for a time step, the CSP considers the concentration level is O for this time step

TABLE 4.1: User inputs, in detail

4.6 Conclusion

This chapter introduces the tool prototype as a Web Application, back-end and front-end.

First a review of the requirements is presented, which leads to architectural and technological choices in the context of this thesis.

Back-end technologies are reviewed, and the choice is motivated. The back-end is a Prolog-coded server, implementing a Web Service and expecting a dedicated JSON file as input. The architectural style on which the server is clearly based upon is the REST style, presented in this chapter.

Front-end technologies are reviewed as well, and the implementation selected is a plain HTML/CSS/JavaScript single page application.

As largely discussed, the production-grade quality attributes are left out of the scope of this prototype – back-end or front-end side –, as the focus of this work is voluntarily kept on the core of the back-end, the constraint logic program. However, several ideas and technologies are introduced in order to guide further development, in the perspective of a in-production application. Those perspectives are reminded in Chapter 8 (p105).

This chapter introduces in fine details the description of the network and modeling task as JSON formatted data. This JSON constitute the interface between any front-end and the back-end.

The next chapters introduce the implementation of the back-end (Chapter 5 (p57)) and the front-end (Chapter 6 (p91)).

Chapter 5

Back-end development

5.1 Introduction

This chapter finely details the back-end implementation.

As expressed in Chapter 4 (p41), the back-end consists mainly in a Web service developed, following REST architectural style. It is coded in Prolog, and does not properly handle production-grade quality attributes. However, it completely integrates the constraint logic programs, at the heart of this thesis' work and contribution.

Firstly, the layered structure is introduced, with explicitly the separation between the controller – from the MVC pattern –, and the constraint logic program, core of the Prolog implementation. Secondly, the CLP is analyzed, and the different steps are detailed. The steps follows the nominal resolution discussed in Chapter 2 (p19), with the association list construction eliciting the variables, then the constraints implementations – from the user, from the modeling followed, or from other biological hypotheses –, and finally the labeling techniques followed.

As several implementations regarding the main modeling constraints and labeling are suggested, the development of a systematic tests framework is detailed, that helps assessing the different methods. The set up and use of the systematic tests framework and observations based on the results obtained are largely discussed. Other functionalities integrated are tested from a high level perspective.

5.2 General overview

There is no real IDE dedicated to Prolog development, and the best tool remains a simple word processing software and a terminal to compile the produced code. While constituting a attractive form of programming, it also imply a need for discipline in the code structure to keep it organized, clean, readable and bottom-line efficient.

Following the layered principle, the source code is split in different files, as visible in the Figure 5.1 (p58). This diagram is organized as the code structure, in layers.

The back-end is organized in different layers, having their own responsibilities.

The first layer consists in the requests handlers in the controller. These handlers are found in `prolog_server.pl` source file. An HTTP request, coming from the client, is received by the server. It contains the JSON inputs data to process, arrives at the controller which, after a first set of verifications, successively calls the predicates required: translation of the JSON input into usable Prolog data – implemented in `json2go.pl` –, resolution using the constraint solver – implemented in `phase0.pl`–, then JSON production – implemented in `json2go.pl`–, and reply to the client.

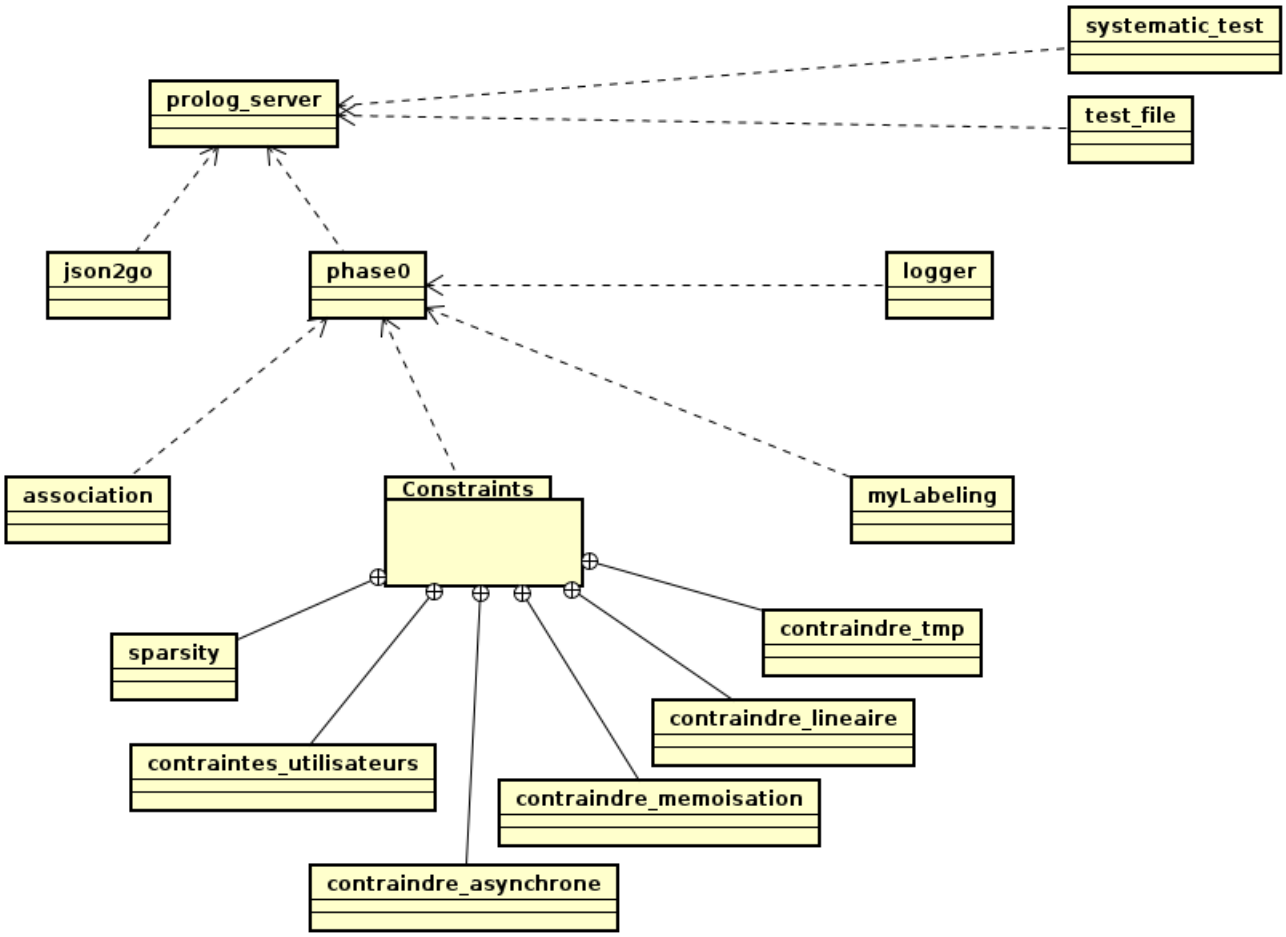

FIGURE 5.1: Layered view of the back-end source code in Prolog

Faithful to the idea of prototyping and bootstrapping the concept, this prototype server remains tiny and limited in the functionalities provided to the user. Corollary, no *view* is generated by the server. That is, there is no HTML code, web page, or visual information that is generated at this level. The *VIEW* part of the infamous MVC pattern, represented in Figure 5.2 (p59), see [42], which can be informally described as all the (visual) interactions with the user, is only handled by the front-end (see Chapter 6 (p91)). This confirms the interest for the REST architectural style selected and the Web Service concept, as explained in Section 4.3 (p42).

The layered split of the code, although quite natural, allows to update the *business part*, the processing of the constraints, without modifying the server. In case of business modification, the controller is impacted only when a change in the interface occurs.

5.3 Controller Layer

The controller part of the MVC pattern is implemented directly in `prolog_server.pl` predicate.

The controllers are considered as the predicates that guide the user requests to the correct Model predicates. In this first implementation, the Web Service being built is rather tiny in regards to the services provided. It would not be the case generally speaking, and having a controller layer separated helps defining new services, preparing in such the extension.

After starting the server with built-in predicate `http_server`, one can access to the web

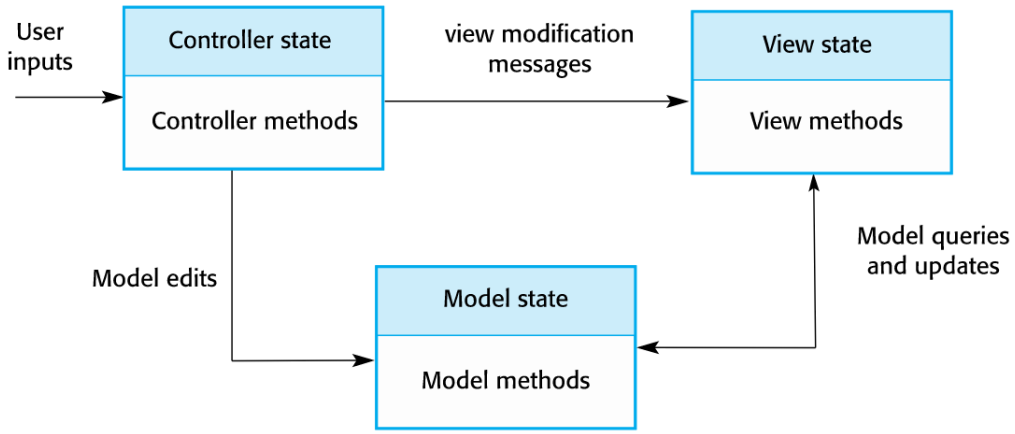

FIGURE 5.2: Schematic view of the MVC pattern

service sending a HTTP POST type request to `/api` handler, which immediately dispatch the request to the `handle_api` controller. This controller verifies first the request type, and eventually the request is taken care of by the POST controller, which calls successively the JSON reading predicate, built-in - hence, the interest of the JSON data exchange file for this implementation -, then the `handle_api_` predicate which will guide through low levels predicates `myTranslate/15` and `multiple_instances/15`, which are detailed in Section 5.4.1 (p60). The built-in `reply_json` reshapes the output of the MODEL to be properly sent, attach the nominal code 201 as HTTP response status, then sends it to the VIEW part, the client (front-end implementation).

It is well *just* a Web Service as fundamentally, from the outside, a JSON data file arrives to the server, and a JSON data file is sent as the response. No HTML page is generated at this point by the server.

```
server(Port):-
        http_server(http_dispatch, [port(Port)]).

:-http_handler('/api', handle_api, []).

handle_api(Request) :-
                option(method(options), Request), !,
                cors_enable(Request, [ methods([post])]),
                format('~n').

handle_api(Request) :-
                option(method(post), Request), !,
                http_read_json(Request, JsonIn),
                setup_call_cleanup(
                        createLog,
                        handle_api_(JsonIn,DictOut),
                        closeLog),
                cors_enable,
                reply_json(DictOut,[status(201)]).

% Modified predicate - details in appendix
handle_api_(JsonIn,Jsons) :-
```

```
        myTranslate/15,
    multiple_instances/15.
```

This constitutes the very base of the server developed. As pointed out in Chapter 4 (p41), an extensive server using more machinery has not been judged useful for this prototyping tool to run, considering the few predicates line that are actually required to access to the CSP, the core of the back-end. Chapter 8 (p105) will however explain why such an *intermediate* server using other technologies could be useful, in a production-grade environment.

5.4 Constraint Logic Program

The *Business* part of the server is dedicated to process the inputs of the user in order to infer *a* GRN, according to what was described in previous sections of this report.

The core of this part starts with the `solve` predicate[1], which really consists in the constraint logic program. This section, taking a top-down approach, details how the program is built, and how technically, the constraints elicited in Section 3.4 (p36) are implemented.

Only the main ideas and the specificities are detailed in this chapter. The curious reader will find all the code in Appendix A (p121).

5.4.1 Preambule: JSON processing

The JSON file such as presented in Section 4.5 (p51) is converted into usable Prolog compounds and terms. Using unification, the conversion is rather straightforward considering the built-in predicate `http_read_json/2`, that takes the HTTP request as input, and outputs the Prolog JSON equivalent. From this equivalent, the conversion to the required terms, considering a few lists manipulation for reshaping, is direct.

As concrete example, hereunder is given an original request JSON file and its Prolog equivalent. With this equivalent, the unification is straightforward.

Original JSON:

```
{"network":[
{"name":"association_list",
"borneMax":10,
"borneMin":0,
"borneEffectOnOthers":5,
"borneEffectOnSelf":0,
"globalThreshold":9,
"steps":2,
"method":"memoisation",
"sparsity":0,
"labeling":"all_ff",
"nSol":1,
"nodes":[
{"label":"gene1","type":"node"},
{"label":"substA","type":"control"}],
"edges":[
{"id":"toChange", "from":"substA", "to":"gene1", "threshold":4,
        "borneEffect":5, "delay":0,"effect":"/"}],
"data":[
```

[1] As concerned about the logic programming, we speak about predicates and not methods or procedures

```
{"node":"substA","step":2,"niveau":7},
{"node":"substA","step":1,"niveau":7},
{"node":"substA","step":0,"niveau":6},
{"node":"gene1","step":0,"niveau":5}]}]}
```

Prolog-equivalent:

```
json([network=[
        json([name=association_list,
        borneMax=10,
        borneMin=0,
        borneEffectOnOthers=5,
        borneEffectOnSelf=0,
        globalThreshold=9,
        steps=2,
        method=memoisation,
        sparsity=0,
        labeling=all_ff,
        nSol=1,
        nodes=[
        json([label=gene1,type=node]),
        json([label=substA,type=control])],
        edges=[
        json([id=toChange,from=substA,to=gene1,
                threshold=4,borneEffect=5,delay=0,effect=(/)])],
        data=[
        json([node=substA,step=2,niveau=7]),
        json([node=substA,step=1,niveau=7]),
        json([node=substA,step=0,niveau=6]),
        json([node=gene1,step=0,niveau=5])]])]])
```

The shaping of the response in JSON happens in two steps: first, the solutions are computed, as a list of Prolog-equivalent JSON terms. The number of solutions depend on the user input, as explained in 4.5 (p51). Those Prolog-equivalent JSON terms are then compiled together with the predicate `build_final_json(+DictOuts,-Jsons)`, where the `DictOuts` is the list of all the solutions found, and `Jsons` is the final response Prolog equivalent. The built-in predicate `reply_json` transform this `Jsons` to send properly the response in standard JSON format.

For the simple network provided, an example of the output with two solutions computed is given in Prolog-equivalent JSON term. This enlightens the respect of the JSON data format structure for both request and response.

```
{"network":[
{"name":"association_list",
"borneMax":10,"borneMin":0,"borneEffectOnOthers":5,"borneEffectOnSelf":0,
"globalThreshold":9,"steps":2,"method":"memoisation",
"sparsity":0,"labeling":"all_ff","nSol":2,
"nodes":[{"label":"gene1","type":"node"},{"label":"substA","type":"control"}],
"edges":[
{"id":"toChange","from":"gene1","to":"substA","threshold":0,
        "borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"substA","to":"gene1","threshold":4,
        "borneEffect":5,"delay":0,"effect":-5}],
"data":[
```

```
{"node":"substA","step":2,"niveau":7},
{"node":"substA","step":1,"niveau":7},
{"node":"substA","step":0,"niveau":6},
{"node":"gene1","step":2,"niveau":0},
{"node":"gene1","step":1,"niveau":0},
{"node":"gene1","step":0,"niveau":5}]},

{"name":"association_list",
"borneMax":10,"borneMin":0,"borneEffectOnOthers":5,"borneEffectOnSelf":0,
"globalThreshold":9,"steps":2,"method":"memoisation",
"sparsity":0,"labeling":"all_ff","nSol":2,
"nodes":[{"label":"gene1","type":"node"},{"label":"substA","type":"control"}],
"edges":[
{"id":"toChange","from":"gene1","to":"substA","threshold":0,
        "borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"substA","to":"gene1","threshold":4,
        "borneEffect":5,"delay":0,"effect":-4}],
"data":[
{"node":"substA","step":2,"niveau":7},
{"node":"substA","step":1,"niveau":7},
{"node":"substA","step":0,"niveau":6},
{"node":"gene1","step":2,"niveau":0},
{"node":"gene1","step":1,"niveau":1},
{"node":"gene1","step":0,"niveau":5}]}]}
```


5.4.2 Program structure

As defined in Section 2.3.5 (p31), the structure follows three steps:

1. Three predicates takes the inputs of the user to build together the data structure that will pass along all the other steps. This is detailed in Section 5.4.3 (p63)

2. The constraints, coming from the user or from the biological model described in Section 3.2 (p33), are detailed in Sections 5.4.5 (p64), 5.4.6 (p65), and 5.4.7 (p69),

3. The labeling, either following standard guidelines or personalized for the problem, is detailed in Section 5.4.8 (p70).

The predicate `solve/13`[2] directly calls `reseau/13`[3]. Its signature is given below. The arguments names are as defined in Section 4.5 (p51). The inputs are noted according to classical Prolog syntax. The output, `Lassoc`, is largely defined in the section related to the association list, see Section 5.4.3 (p63).

```
reseau(+Method, +Sparsity, +Labeling, +Nodes, +Edges, +Data,
        +BorneMinNiveau, +BorneMaxNiveau, +BorneEffectOnOthers,
        +BorneEffectOnSelf, +GlobalThreshold, +Steps, -Lassoc).
```

[2]This is the notation as used in the literature, where the amount of inputs or outputs are denoted as an integer following a slash. This notation will be used in the report

[3]The reason of these two predicates being pretty much the same thing is historical: another *branch* was at first foreseen at this node, but left out later on. This has zero impact on the rest of the codes.

5.4.3 Association List

The association list, noted `Lassoc` in the predicates, is the structure that gathers all the variables on which constraints are applied. Considering the context, `Lassoc` includes three kinds of predicates to link the variables together:

- `niveau/4`, that links a concentration level to a given time step, for a particular gene.
- `gen_gen/7`, that describes an edge of a GRN. It is considered that the GRN is a complete graph, hence all the potential edges are automatically created. All the genes are connected to all other genes by an edge (unique, at this point) represented by `gen_gen`. The `gen_gen` predicate explicits an *ID*[4], the *source* and *target* genes, the *threshold* of this edge within the meaning of Section 1.3.5 (p11), the *maximal bound* that this edge can have, the *delay* within the meaning of Section 1.3.5 (p11) and eventually the effect the source gene has on the target gene. In the code, the predicate is usually noted `gen_gen(ID, GeneFrom,GeneTo,Threshold,BorneEffect, Delay, Effect)`,
- `self/2`, that is related to the effect that a gene has on itself, and is noted `self(Gene, Effect)`. This relation is not used in this first version of the system, although the architecture and interface has already been foreseen.

The list `Lassoc` is built based on the main inputs from the user : the list of genes in the system, the amount of discrete steps during which the GRN is considered, and the global bounds or thresholds. At this time, the particular inputs associated to a specific gene or edge are not considered. For instance, if the user selected a maximum threshold of 10 for the GRN under consideration, but knows also that for an edge specified from *geneA* to *geneB*, the threshold is 6, this information is not yet taken into account.

The domain of each variable is limited at that time according to these global inputs:

- the `Threshold` is bound in $[1, \texttt{GlobalThreshold}]$,
- the `Delay` is bound in $[0, 1000 * N]$ where N is the number of steps[5],
- the `BorneEffect` is bound in $[0..\texttt{BorneEffectOnOthers}]$
- the `Effect` is bound in $[-\texttt{BorneEffect}, +\texttt{BorneEffect}]$,

Three predicates build each part of `Lassoc`, which are then appended to produce the complete structure. This structure is abundantly used in the rest of the program.

Example. As a concrete example, taking the simple JSON input file from Section 5.4.1 (p60), the corresponding association list is given hereunder. The graphical representation of this network (nodes and interactions) is given in Figure 5.3 (p64).

```
Lassoc = [
        niveau(control,substA,2,Var1),
        niveau(control,substA,1,Var2),
        niveau(control,substA,0,Var3),
        niveau(node,gene1,2,Var4),
        niveau(node,gene1,1,Var5),
        niveau(node,gene1,0,Var6),
```

[4]The ID is currently not used at all in the first version, and is automatically changed to `"toChange"` to emphasize it is not taken care of. This behavior should change in a future version

[5]The upper bound is set up purely arbitrarily

```
gen_gen(toChange,gene1,substA,0,0,0,0),
gen_gen(toChange,substA,gene1,Var7,Var8,Var9,Var10),
self(substA,0),
self(gene1,0)]
```

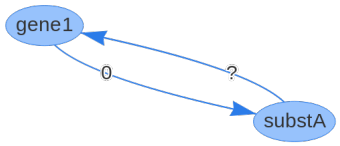

FIGURE 5.3: Graphical representation of the network - `Lassoc` example

5.4.4 Constraints - introduction

Different constraints need to be modeled in order for the solver to find an appropriate solution, respectful of Lucy's domain. The first one is related to the user inputs as described in Section 4.5 (p51), the second to the GRN itself, as explained in 3.4 (p36). It has been decided to allow two substances types, as defined in Section 4.5 (p51): `node` and `control`. If node represents a classical gene, as usually expressed in this work, a `control` substance can be whatever entity which concentration is not subject to `gen_gen` interactions, while still participates to the regulation. The concentration levels for each time step need to be provided, or is considered to be 0 by default. It allows to introduce completely controlled entity in the domain, inducing external `gen_gen` interactions that applies on the `node` entities, hence providing an additional degree of freedom for the modeling and hypothesis tests.

The sparsity constraint, inherent to the application domain, is also provided.

5.4.5 User constraints

As elicited in Section 4.5 (p51), the user has the possibility to give several different inputs to the system. The fact that those inputs come from experimental data, observations, other studies, or assumptions is irrelevant in the context of the tool itself.

Those inputs from the user are directly linked to `Lassoc`, and all the information added will eventually be linked to a reduction of the domain of one of the variables contained in `Lassoc`. More specifically:

- if a specific threshold or delay is given for an edge, `Lassoc` is modified accordingly,
- if a `BorneEffect` is given for an edge, `Lassoc` is modified, and the `Effect` subject to this `BorneEffect` has its domain reduced,
- if an `Effect` is given, Lassoc is modified accordingly,
- if a data is added, specifying a level of a gene at a specific time step, `Lassoc` is modified accordingly

The back-end handles the fact that an edge may not be fully known, leading to incomplete or partial information. Typically, it could concern the `Threshold` or `Delay`, for instance. Yet, the information sent via the JSON file in the request is integrated within `Lassoc`. This was introduced in Section 4.5 (p51).

5.4.6 Constraints on the Gene expression Level

As indicated in the modeling section, see Section 3.4 (p36), the gene expression levels, marked `X`, are computed based on Equations 3.1 (p36). Those equations can be understood as constraints on the variables associated with these gene expression levels.

The amount of variables to constraints is important : $(N+1)$ [time steps] $\cdot n$ [genes][6]. To set the constraint, each expression level variable needs:

- the n variables associated with a previous gene expression level,
- the $n-1$ variables associated to the `Threshold` of each potential interaction,
- the $n-1$ variables associated with the `Delay` of each potential interaction,
- the $n-1$ variables associated with the `Effect` of each potential interaction.

Those relations being the very core of the constraint program, it is worth spending time on them. There are multiple ways of implementing these constraints, and three of them will be presented for the synchronous state.

As presented in Chapter 2 (p19) and Section 2.2.4 (p23), the constraints should be expressed easily. This is the idea of the first method followed, called `lineaire`.

5.4.6.1 Constraining using `lineaire`

This implementation is the closest to the common interpretation of the sum represented by Equations 3.1 (p36). It consists mainly in recursive predicate over `Lassoc`, and includes the following major steps if the element under concern is `niveau(Gene,Temps,Var)` with `Temps>0`:

1. Find all the `Expressors`, meaning all the genes for which an association `gen_gen(_, Expressor, Gene, _, _, _, _)` exists. The classical (and ISO function) `findall/3` predicate is used to that end:
   `findall(Expr, member(gen_gen(_,Expr,Gene,_,_,_,_), Lassoc), Exprs)`,
2. Convert this genes list to actual *rate* list, or effect list, using the predicate `convertGeneListToRateList/5`, explained below,
3. Sum the rate list obtained in order to get the `Contribution`. This uses the built-in predicate `sum/4`
4. Apply the `boundaries/4` predicate, to make sure the value computed stays in the interval required. The details are given below.

[6] +1 because of the start at 0

convertGeneToRateList/5 Let's take a target gene g and the expressors list found at the previous step being $[a]$. The expression level to constrain is $\texttt{X}_g$ at time `Temps`. As input, `convertGeneToRateList/5` takes the list of expressors found previously = $[a]$. Recursively, it converts this expressors list to a list of `Effect` according to the $\texttt{X}_a$ being above or below the $\texttt{Threshold}_{ab}$, as indicated in Equations 3.1 (p36). For each gene of the Expressors list, there are three possibilities for this conversion:

- **case1**: the effect of the interaction $a \rightarrow g$ is known, and is 0. In that case, no matter the concentration level of a, the effect to consider for the `Contribution` of X_g is 0,
- **case2**: $\texttt{X}_a$ at the time `Temps - Delay - 1` is smaller that $\texttt{Threshold}_{ag}$, or the time `Temps - Delay - 1` is negative. Then the effect to consider for the `Contribution` of X_g is also 0,
- **case3**: $\texttt{X}_a$ at the time `Temps - Delay - 1` is greater than or equal to $\texttt{Threshold}_{ag}$. Then the effect to consider for the `Contribution` of X_g is $\texttt{Effect}_{ag}$.

Basically, this distinction follows directly Equations 3.1 (p36). The recursion ends up with a list of `Effect`'s. The very details of the implementation are given in Appendix A (p121). The sum gives the `Contribution` to add to the concentration level of the previous time, in order to get the concentration level at the present time.

boundaries/4 This predicate is built in such a way that does not introduce choice point (only one clause), to limit useless backtracking during the search:

```
boundaries(In,Out,Min,Max):-
        V1 #= min(In,Max),
        Out #= max(V1,Min).
```

5.4.6.2 Constraining using `tmp` method

As indicated by the denomination, this method originally was a test that actually proves itself of some interest. The idea is to split the information collection from the threshold processing and, by doing so, limit the possible backtracking required by the threshold modeling, see Section 3.4 (p36).

At first, the method is similar to the `lineaire` method as the main predicate is recursively applied over `Lassoc`, with the same main steps. The two methods differ however in the way the information to compute `Contribution` are obtained. In the `tmp` method, first, the `Effect`'s of interactions are compiled in a list, along with the corresponding `Niveau`'s and `Threshold`'s. Second, the boolean values are obtained based on the `Niveau`'s and `Threshold`'s.

Three predicates are responsible for the computation of the `Contribution` :

```
getAllInfo(+Expressors,+Gene,+Temps,+Lassoc,-Niveaux,-Thresholds,-Effects),
apply_predicate_list(+Niveaux,+Thresholds,-BooleanValues),
scalar_product(+BooleanValues,+Effects,#=,+Contribution),
```

`scalar_product` is a built-in predicate, and `getAllInfo/7` uses pure recursion over `Lassoc` to gather the relevant information. The details are left in Appendix A (p121).

apply_predicate_list/3 A dedicated predicate is used to apply, over lists, the threshold idea.

```
apply_predicate_list(Ls1,Ls2,Ls3):-
        maplist(gt_threshold,Ls1, Ls2, Ls3).

gt_threshold(L1,L2,L3) :-
        L1 #>= L2,
        L3 is 1.
gt_threshold(L1,L2,L3) :-
        L1 #< L2,
        L3 is 0.
```

It takes benefit from the `maplist` built-in predicate, optimized at low-level to faster apply a predicate over lists. `gt_threshold` is a homemade implementation : it is reviewed in the next method presented, `memoisation`

5.4.6.3 Constraining using `memoisation` method

Inspired by the previous method, constraining using `contraindre_memoisation` splits completely the part of the work dedicated to grabbing information with the constraints application. This method does not support the `Delays` introduced by the user.

Information formatting. The first step is to get the information required to - later on - apply the constraints and to format it in a usable structure. This is done in one recursion of the association list `Lassoc`. Considering the modeling introduced, the information is, for each expression level `Var`, its previous value `VarPrev`, then the list of `Niveau`, `Threshold`, `Effect` of all genes that can contribute to `Var`, according to Equations 3.1 (p36).

A concrete example is given here under, considering the very basic network given.

```
{"network":[
        {
        "name":"memoisation_explained",
        "borneMax":10,
        "borneMin":0,
        "borneEffectOnOthers":5,
        "borneEffectOnSelf":0,
        "globalThreshold":9,
        "steps":2,
        "method":"memoisation",
        "nodes":[
                {"label":"gene1","type":"node"},
                {"label":"gene2","type":"node"},
                {"label":"gene3","type":"node"}],
        "edges":[],
        "data":[]
        }
]}
```

This network has for association list :

```
Lassoc = [
niveau(node,gene3,2,X32),niveau(node,gene3,1,X31),niveau(node,gene3,0,X30),
niveau(node,gene2,2,X22),niveau(node,gene2,1,X21),niveau(node,gene2,0,X20),
niveau(node,gene1,2,X12),niveau(node,gene1,1,X11),niveau(node,gene1,0,X10),
gen_gen(toChange,gene1,gene2,Th12,_,_,Eff12),
gen_gen(toChange,gene2,gene1,Th21,_,_,Eff21),
```

```
gen_gen(toChange,gene1,gene3,Th13,_,_,Eff13),
gen_gen(toChange,gene3,gene1,Th31,_,_,Eff31),
gen_gen(toChange,gene2,gene3,Th23,_,_,Eff23),
gen_gen(toChange,gene3,gene2,Th32,_,_,Eff32),
self(gene3,0),self(gene2,0),self(gene1,0)]
```

And the resulting structure is :

```
[
[X32,X31]-[[X11,X21],[Th13,Th23],[Eff13,Eff23]],
[X31,X30]-[[X10,X20],[Th13,Th23],[Eff13,Eff23]],
[X22,X21]-[[X11,X31],[Th12,Th32],[Eff12,Eff32]],
[X21,X20]-[[X10,X30],[Th12,Th32],[Eff12,Eff32]],
[X12,X11]-[[X21,X31],[Th21,Th31],[Eff21,Eff31]],
[X11,X10]-[[X20,X30],[Th21,Th31],[Eff21,Eff31]]
]
```

Considering the first element of the matrix structure, all the information are accessible to later on constraint `X32`, the expression level of *gene3* for the time step *2*. This constraint, or equation, comes directly from Equations 3.1 (p36) :

$$\texttt{X32} = \texttt{X31} + s(\texttt{X11}, \texttt{Th13}) \cdot \texttt{Eff13} + s(\texttt{X21}, \texttt{Th23}) \cdot \texttt{Eff23} \tag{5.1}$$

At first, this part was built using extensively `findall/3` and `bagof/3`, built-in predicates, to take advantages of the low-level optimizations. However, after extensive tests, it appears that this route was quite a dead-end, at least for several configuration : `findall/3` and `bagof/3`, handle quite badly the constraints propagation. Because of user inputs and association list construction, all the variables are already constrained, and the predicates'performances dramatically fall as the amount of variables increase. This was shared also in [47]. As a result, a conventional approach using recursion was preferred, to limit the (unknown) side effects. A benefit of using classical recursion, on the other side, is the portability of the code with respect to Prolog implementation.

Constraint application. The constraints are applied recursively on this structure only. Provided entries from the user, as defined in Section 5.4.5 (p64), the search will backtrack on this part only, where are the latest choice points related to the boolean function $s(x, \theta)$. In order to (try to) optimize the performances, the built-in predicate `zcompare/3` is used for the $s()$ function[7], wrapped in a `myzcompare/4` that outputs 1 or 0 depending on the concentration level being higher or lower than the threshold. The `myzcompare/4` is applied on the lists of `Niveau` and `Threshold` using the built-in predicate `maplist`, once again in order to optimize the execution. Eventually, the value obtained following Equation 5.1 (p68) in bounded by the classical `boundaries/4` explained above.

```
apply_constraint(_,_,[]).
apply_constraint(BorneMinNiveau,BorneMaxNiveau,Matrix):-
        Matrix = [[Var,VarPrev]-[Niveaux,Thresholds,Effects]|Tail],
        maplist(myzcompare,Relations,Niveaux,Thresholds,TrueVal),
        scalar_product(TrueVal, Effects, #=, Contribution),
        VarTmp #= VarPrev+Contribution,
        boundaries(VarTmp,Var,BorneMinNiveau,BorneMaxNiveau),
        apply_constraint(BorneMinNiveau,BorneMaxNiveau,Tail).
```

[7] instead of the `gt_threshold/3` used in the `tmp` method

5.4.6.4 Asynchronous

The asynchronous case, as discussed in Section 3.4.1 (p36) follows four assumptions in this work. The major difference is related to the `Delay` variable, which semantic changes : the `Delay`'s are pairwise all different, and the value of the delay $a \rightarrow g$ is directly linked to the application of the interaction $a \rightarrow g$ on g. It is considered (**H3**) that the application of the interactions respect a fixed sequence.

Following the four assumptions, the steps of the implementation are:

1. Get all the `Delay`'s variables and limit their domain to $[0, \mathit{NumberOfDelays}]$,
2. Impose the global constraint (see Section 2.2.2.2 (p22)) that all `Delay`'s are different,
3. Apply recursively, over the time steps, the constraint on the expression level:
   - Find the applicable interaction for the time step under consideration,
   - Compute and apply the `Contribution` to the expression level of the target gene using the very same way as described in the `linear` method,
   - For all the other genes - not target of the applied interaction -, constraint the expression level to be the same as the previous time step.

5.4.7 Sparsity

Sparsity denotes the limitation of the amount of interactions that have an effect different from 0. As explained in Section 3.4 (p36), it has a real biological interest, and for the CLP itself, it can help limit the global amount of solutions that fulfills the constraints : it may not decrease the time required to find the first solution, but it will limit the total amount of solutions possible.

`sparsity(+NbreMaxEffect,+BorneEffectOnOthers,+Lassoc)` imposes that no more than `NbreMaxEffect Effect` are different from 0 in `Lassoc`. The implementation takes advantage of a common global constraint (see Section 2.2.2.2 (p22)), `global_cardinality/2`, see [23]:

" `global_cardinality(+Vs, +Pairs):` Vs is a list of finite domain variables, Pairs is a list of Key-Num pairs, where Key is an integer and Num is a finite domain variable. The constraint holds iff each V in Vs is equal to some key, and for each Key-Num pair in Pairs, the number of occurrences of Key in Vs is Num."

- `+Vs` is defined as the list of all `Effect` from `Lassoc`, that is straightforward to get,
- `Pairs` is the list of `Key-Num`, where `Key` is a possible value that `Effect` can take, and `Num` is the number of occurrences of this value in the list of `Effect`'s.

At this point, a concrete example may be useful. Consider the association list :

```
Lassoc = [ ...
gen_gen(_,gene1,gene2,_,_,_,Eff12),
gen_gen(_,gene2,gene1,_,_,_,Eff21),
gen_gen(_,gene1,gene3,_,_,_,Eff13),
gen_gen(_,gene3,gene1,_,_,_,Eff31),
gen_gen(_,gene2,gene3,_,_,_,Eff23),
gen_gen(_,gene3,gene2,_,_,_,Eff32),
... ]
```

Then,

```
?- sparsity(2, 5, Lassoc).
List_of_Effects = [Eff12,Eff21,Eff13,Eff31,Eff23,Eff32],
Pairs_Key-Num = [0-V1,-5-V2,-4-V3,-3-V4,-2-V5,-1-V6,1-V7,2-V8,3-V9,4-V10,5-V11],
[V2,V3,V4,V5,V6,V7,V8,V9,V10,V11] ins 0..2,
V2+V3+V4+V5+V6+V7+V8+V9+V10+V11 #=<2,
V1 in 4..6.
```

where $\texttt{V}_i$ are the occurrences of the value denoted by the `Key` in the list of Effects. Except for the variable associated to 0, all the domains of $\texttt{V}_i$ are - considering the worst case where all the `Effect`'s would have only one value - $[0, \texttt{NbreMaxEffect}]$, which is $[0, 2]$ in the example shown. The sum of all $\texttt{V}_i$ should be less than or equal to `NbreMaxEffect`. The variable remembering the number of occurrences of the value 0 among the `Effect`, variable `V1`, has a domain compatible with the `NbreMaxEffects`, so $[4, 6]$ in the shown example.

Applying the constraint `global_cardinality` makes the link between the `Effect` variables, and the $\texttt{V}_i$ variables.

5.4.8 Labeling

The labeling phase is of particular relevance considering the application domain. As hinted in Section 3.5 (p38), there is quite a chance that the problem will be under-constrained, leading to several - even lots of - different acceptable solutions.

Among this set of solutions, a few are more relevant than the others. The main goal of the implementation of the labeling is to give first the most relevant possibilities - while still offer the opportunity to observe the others.

The central predicate, `labeling/2` for the implementation of the labeling technique is given, with pre-defined many possible options, by SWI-PROLOG. The first argument is a list (possibly empty) of options, and the second argument is the list of variables to be labeled. The list of variables is obtained by recursion on `Lassoc`, according to different predicate for labeling the whole bunch of variables, or specific variable types only, as discussed in Section 3.5 (p38).

`Labeling/2` is always complete, always terminates, and yields no redundant solutions, as elicited in [59]. It is strongly advised to use the native implementation[8] unless required.

Four different variables ordering are implemented and tested:

1. Simple ordering, by default. As announced in the documentation [59], the default consists in labeling the variables in the order they occur,

2. First Fail ordering. The variables are sorted before labeling. The list of variables contains the five types of variables introduced in the modeling (`Niveau`, `Effect`, `Threshold`, `Delay`, `BorneEffect`),

3. Home-made ordering: The labeling follows this ordering:

   (a) `Niveau` variables

   (b) `Effect` variables

   (c) `BorneEffect` variables

   (d) `Threshold` variables

   (e) `Delay` variables

[8] Note that the implementation itself can be found in [67] - helping for portability purpose, if a reimplementation is required

This ordering is arbitrary, although labeling first the variables that could most likely induce a failure and that are the most constrained.

4. Reverse Home-made ordering : the labeling follows the exact reverse order of the previous ordering. The intent is to show the impact that the ordering can have on the performances.

Along with changing the variables ordering, as introduced in Section 5.4.8 (p70), the value for each variable can be set to integrate real world consideration.

As a reference, the default value ordering for three variables is given here under, and act as reference for the following discussion:

```
Vars = [V1,V2,V3], Vars ins -5..5, labeling([],Vars).
Vars = [-5, -5, -5]; Vars = [-5, -5, -4]; Vars = [-5, -5, -3];
Vars = [-5, -5, -2]; Vars = [-5, -5, -1]; Vars = [-5, -5, 0];
Vars = [-5, -5, 1]; Vars = [-5, -5, 2];Vars = [-5, -5, 3];
Vars = [-5, -5, 4]; Vars = [-5, -5, 5]; Vars = [-5, -4, -5];
Vars = [-5, -4, -4]; Vars = [-5, -4, -3]; ...
```

The main goal of using a different value ordering, at least in this work, is not specifically to have a more efficient labeling, but rather to find the most relevant solutions first. The measures that have been implemented are :

`Effect` values ordering. The `Effect` are labeled as if the statistical distribution of the `Effect`'s values would follow a Gaussian, centered at 0.
For instance, let's imagine the variables `[Eff1, Eff2, Eff3]` represent effects that need to be labeled, and the domain of all effects is $[-5, ..., 5]$. The values that will be tried out first are given by:

```
[Eff1,Eff2,Eff3]=
[0, 0, 0];
[-1, 0, 0];[0, 0, 1];[0, 1, 0];[0, 0, -1];[1, 0, 0];
[0, 0, 1];[1, 0, 0];[0, 1, 0];[1, 0, 0];[0, 0, 1];
[-2, 0, 0];[0, 0, 2];[-1, -1, 0];[0, 1, 1];[-1, 0, -1];
[1, 0, 1];[-1, 0, 1];[1, 0, 1];[-1, 1, 0];[0, 1, 1];
[0, -2, 0];[0, 2, 0];[0, -1, -1];[1, 1, 0];[0, -1, 1];
[1, 1, 0];[0, 0, -2];[2, 0, 0];[0, 0, 2];[2, 0, 0];
[0, 1, -1];[1, 1, 0];[0, 1, 1];[0, 2, 0];[1, -1, 0]; ...
```

The ordering is clearly really different, favoring first values close (either positive or negative) to 0. To implement this ordering, the native `labeling/2` predicate is used, with specific options set: the minimization of the sum of all the effects. During the labeling, an additional constraint (as introduced in Section 2.2.1 (p20)) is therefore added between all the `Effect` variables: the sum of the absolute values of all the effects labeled shall be minimum.

`Threshold` and `BorneEffect` values ordering. After the `Niveau` and the `Effect` variables are labeled, the values assigned to the `Threshold` and `BorneEffect` variables, as long as they respect the constraints, may appear less interesting. As a possibility, in order to remove irrelevant checkpoint, an option could be proposed to the user to directly assign the most relevant bound of the acceptable domain. Assigning the value would decrease the number of choices and, using the backtracking, only a new `Effect` or `Niveau` value given would induce a change in the `Threshold` or `BorneEffect`. This opportunity is implemented also during the labeling phase. As for all the code, the implementation is given in Appendix A (p121).

5.5 Systematic tests

In the previous sections, the development of different methods for applying constraints – specifically for the synchronous case – and for the labeling have been introduced.

As explained in Chapter 2 (p19), different modeling and labeling methods may have an impact on the performance of the search space, and lead of different execution times. This previous chapter also indicated that many real-world problems were to be better solved using dedicated heuristic, to cut down the time required for the resolution considering the usual combinatorial nature of those problems. Analyzing the performances of the methods developed is a first step towards these heuristics.

In order to ease the comparison between the different alternatives developed, a systematic test framework has been built. Overall, it serves as an integration tests framework, where the user input, nominally coming via an HTTP POST request in JSON, as explained in Section 4.3 (p42), is replaced by a local JSON file.

Thanks to the systematic property, a massive amount of tests can be played, allowing to assess as thoroughly as possible the behavior of the method under test.

Synchronous methods for applying constraints. In the context of the three different constraints application methods, the goals are:

1. To confirm that the different methods end up with compatible results. Based on the same modeling hypothesis, only the implementation is supposed to change. The ultimate result of the execution of an input file should not be different using a method or another. This will give an hint on the ***correctness*** of the implementation : if a method gives a different - and not explicable - result than the others, it's likely there is an implementation error, while if the three different methods give the same result, it is likely the implementation is correct - the result of the test depending then on the correctness of the modeling itself.

2. To assess the **performances** of the different methods that are coded differently. Regarding this phase, the less interesting case would be to observe the exact same performances for the three tested methods. Indeed, it would imply that the differences in the implementation have zero impact on the global execution. On the contrary, if differences are enlightened, it could be a hint that one method should preferably be used in a specific case, while another performs better in other conditions. That is, having a notable differentiation in the methods performances (provided it is not the same method that outperforms for all test files), indicate the opportunity for a heuristic development.

Labeling. In the context of labeling, systematic tests allow a comparison of the performances of the labeling algorithm applied – without forgetting that the labeling goals are also to provide the most relevant solutions first.

The global idea of this systematic framework is to generate multiple (realistic) test files, execute them in a row, and log the relevant information for further processing.

5.5.1 Creation of the tests files

In order to run systematic tests, a set of relevant inputs files, representing networks in JSON format, are required. These inputs files will be used, as expressed in the introduction, to assess the performances of the constraints methods (for synchronous models), of the labeling, with the hope that they help providing an heuristic based on the results, or at least give an idea of the behavior of the system according to the inputs given.

A test file generator script has been written in Python to generate these input files. They should ideally sweep a large part of the different parameters that can have an impact. Precisely, the major impacting parameters and their value in the test files generated are:

- the method for constraints application. As a reminder, those methods are the `lineaire`, `tmp`, and `memoisation`, as fully described in Section 5.4 (p60).
- the number of nodes. In the context of this work, this number varies between 2, 3 and 4,
- the number of edges provided, which varies between 0 and twice the amount of nodes, by step of 2,
- the number of data provided, which can vary by skipping 0, 1, 2 or 3 data from the ground truth, which is complete (full edges, full data),
- the number of time steps, which varies from 2 to 8, by step of 2.

The intervals chosen for the parameters are mainly defined to optimize the time spent on the tests, while still obtaining relevant information for comparison. Increasing the value for a parameter (for instance, adding a node) results in an increase of the complexity, hence increasing the timing (and memory) required for the test.

To build a JSON input file, the Python script contains a database of a ground truth, a specific case that has been manually built so that the complete information is mastered. A graphical representation of the ground truth network is given in Figure 5.4 (p73), and the result of its concentration levels is given in Figure 5.5 (p74). The corresponding JSON version is given in Appendix A.6.1 (p148). The graphical evolution of the concentration levels comes from a print screen of the front-end built to ease the development. More information on the front-end is provided in Chapter 6 (p91).

The name of the input files generated gathers the values of the different parameters set, so that the information can be later used by a dedicated parser (see Appendix A.6.5 (p156)).

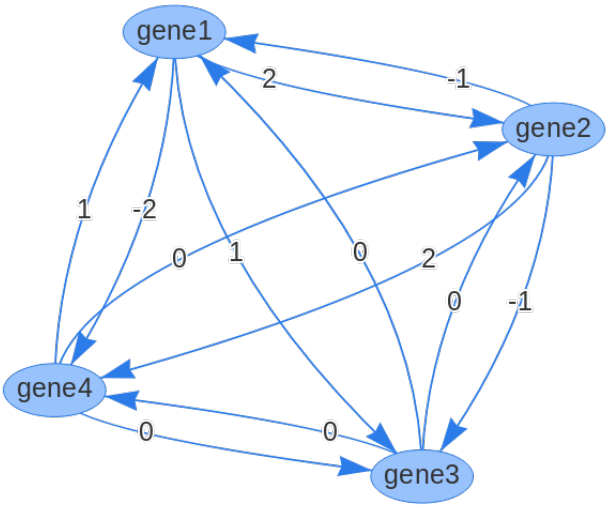

FIGURE 5.4: Ground Truth used for test generation - Graphical representation of the network

In the end, the Python script generates the test files by passing through nested `for` loops in order to have a file generated for all the combinations of the parameters mentioned above.

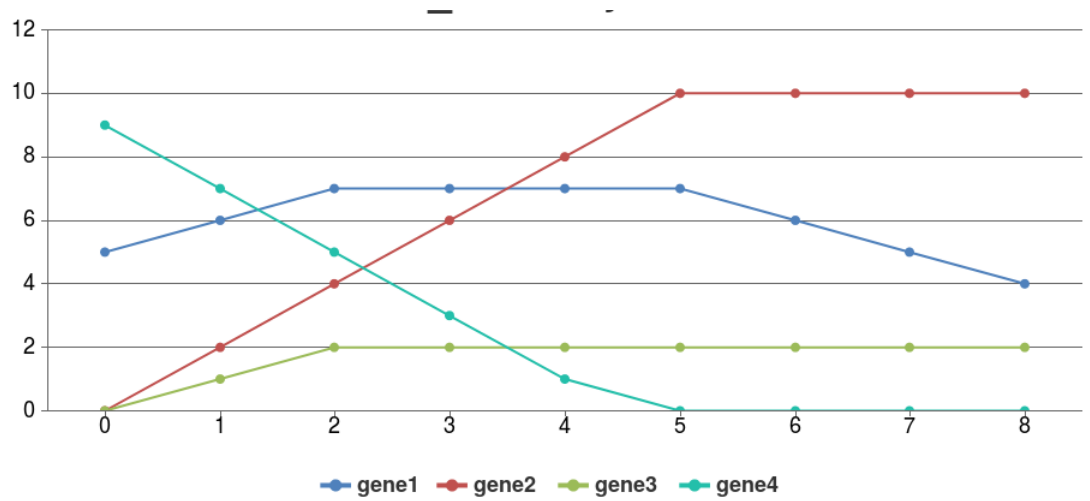

FIGURE 5.5: Ground Truth used for test generation - Concentration levels evolution over 8 steps of time

Relevance of test outcome. As all the files are generated from a ground truth by selecting specific data, the validity of the ground truth is not a guarantee of the validity of all the test files created. Another way to express this is that the fact that the ground truth fulfills the constraints considering Chapter 3 (p33) – and can therefore be called valid GRN – does not mean that all the test files generated by selecting part of the data, edges, nodes, ... from the ground truth will lead to an acceptable GRN. Several files lead to inconsistent constraints and no solution.

However, depending what is at stakes, the fact a test file leads to a positive outcome (inputs can be inferred to a valid GRN) or not (inputs cannot lead to a valid GRN) is not particularly relevant: attesting an input does not lead to a solution is part of the work that the tool (and the constraints) should accomplish. Moreover, a `false` result, indicating that in the whole search space, there is no combination that leads to a GRN considering the inputs, implies often digging a lot, hence taking more time, than finding a `true` result. Rather than the outcome, the timing of the processing, and specifically compared to other methods timing, is the most relevant information of such systematic tests.

5.5.2 Systematic test framework

The Prolog predicate `systematic_test/2` is at the top of this framework. The first argument consists in a term indicating which test list should be played, based on the constraint application method, and the second is the name of the log file that will be saved at the end of the execution. After retrieving the proper list of test to play according to the first argument, `systematic_test` calls recursively the `test/1` predicate, giving the name of the test to play as argument. The call to `test` is wrapped to cover the logging and the timeout cases. When called, `test` applies the exact same steps as a nominal execution would: the input translation to Prolog usable terms, the solving tasks, then the production of the JSON output.

The implementation is given in Appendix A (p121).

5.5.3 Synchronous methods assessment

Section 5.4 (p60) introduced three different methods for applying the constraints that are assessed in this section.

Practically, thanks to the tests generator script described in Section 5.5.1 (p72), 144 test cases covering many different configurations have been generated for each method. Using the

systematic_test predicate, those tests can be played, and the results logged in the indicated logfile, locally. This log file includes the results of all the tests performed, and the time spent or a timeout (considering a time limit given). Another Python script parses the results logged, produces a more readable file and reshapes the results in order to ease the comparison.

This Python script:

- open the logfile provided,
- read the logfile line after line, and for each line:
  1. retrieve all the information logged,
  2. build a matrix with the results.
- print the matrix in a more readable file called `summary.txt`,

The first lines of a logfile before and after reshaping are given here under - the complete file is given in Appendix B (p169).

```
Before:

network_test_memoisation_steps8_nodes3_edges4_data3.json,1534154490.9839895,FAIL,0.010194063186645508,[s]
network_test_tmp_steps4_nodes3_edges2_data3.json,1534154490.9942567,timeout,inf,[s]
network_test_tmp_steps2_nodes4_edges4_data0.json,1534154491.9945407,OK,0.006990194320678711,[s]
network_test_memoisation_steps6_nodes3_edges4_data2.json,1534154492.0015447,FAIL,0.002248525619506836,[s]
network_test_tmp_steps4_nodes3_edges4_data0.json,1534154492.003824,FAIL,0.003116130828857422,[s]

After:

|    TEST    | LIN ([ms]) | TMP ([ms]) | MEM ([ms]) |
|  2-2-0-0   |   0(2.0)   |   0(2.0)   |   0(6.0)   |
|  4-2-2-3   |   0(8.0)   |  0(15.0)   |  0(32.0)   |
|  4-3-0-0   |  0(27.0)   |  0(23.0)   |  0(596.0)  |
|  4-3-0-1   |  1(638.0)  |   /(inf)   |   /(inf)   |
|  4-3-0-2   | 1(1813.0)  |  1(16.0)   |   1(8.0)   |
```

In order to reproduce the experiment and the annexed results, the labeling option for the synchronous tests was the `all_ff`, first fail without specific value ordering. The edges given contains the `Effect`, and the `Delay`, but not the `Threshold` nor the `BorneEffect`.

5.5.3.1 Reading the logfile

After the Python logging parser has been run, the output is reshaped in a humanly readable format, as a table containing the results and the time elapsed for the execution. The following indications may help the reader to get used to the formalism:

- The test name `X-Y-Z-D` explicit the information contained in the input file: X is the number of steps, Y is the number of nodes (only type used was `"node"`, no `"control"` were introduced), Z is the number of edges, and finally D, the data missed. D is the number of data that are skipped between all the data picked from the ground truth list,
- LIN denotes the result running the test inputs with the `lineaire` method. TMP and MEM stand for `tmp` and `memoisation` methods,
- The 0 result denotes a `fail`, a 1 denotes a `true`, and a `/` denotes a timeout (hence, the inf as time),
- The times, in brackets, are given in ms,

5.5.3.2 Tests run

In total, 432 tests are run in a row, 144 for each method. While limiting the execution time to a few seconds, considering the limited resources, the expectation beforehand is to observe the behavior detailed in introduction of this section: the results for all methods should be consistent as they are modeling the same real-world problem, and – hopefully – the methods don't have the exact same efficiency depending on the inputs.

5.5.3.3 Observations

At the end of the tests run, several major observations can be done, reading the processed logfiles.

To make those observations, it may be easier to extract the most relevant information, and to remove for instance all the test cases where either all the methods found the answer within the same time frame, or where no methods could conclude to a result. Such an extract is provided in the Figure 5.6 (p77).

- The very first thing to note is the matching of the results between the three different implementations. Although the way the constraints are applied are differently implemented, the outputs always agree when the test has terminated. This is a very positive indication regarding the implementation itself.
- As expected, the amount of unresolved situations increase as the complexity increase. Considering the run of all tests, a timeout was set after $1[s]$ or $5[s]$. One could imagine much longer experiments and hopefully see the proportion of unresolved cases decrease.
- If all data are all given (as a reminder, data defines a specific `Niveau` for a node, at a given time step), at least one method can find a solution. This is interesting in the context of the GRN inference, as often, the data - although requiring normalization and noise filtration - can be known,
- The second really interesting aspect is the differentiation between the methods. On the extract below, an extra column has been added, manually, to express the most efficient method (based on the initials). As one can see, all the methods are, at some point, the best for specific test cases. This results is very promising regarding the elaboration of a heuristic,
- Overall, the memoisation method seems to be slower than the two others, that are quite similar for many cases,
- `memoisation` seems to be particularly better adapted to situations where the edges are already known, or foreboded.

5.5.3.4 Conclusion

The automatic run of complete cases, and the test file generations based on the ground truth have revealed themselves as a major help on the analysis of the three different methods, indicating the relevance of the systematic tests framework. The observations made would of course need to be refined. Also, still using the framework, extended test cases based on multiple ground truth and more resources could guide towards the appropriate heuristic, and towards designing an expert system capable of deciding the most appropriate method to apply, considering the user inputs.

TEST	LIN ([ms])	TMP ([ms])	MEM ([ms])		TEST	LIN ([ms])	TMP ([ms])	MEM ([ms])	
2-3-0-0	0(17.0)	0(12.0)	0(273.0)	LT	4-4-2-2	/(inf)	1(19.0)	1(17.0)	TM !!
2-3-0-1	1(24.0)	1(5.0)	1(4.0)	M	4-4-2-3	/(inf)	1(134.0)	1(57.0)	TM !!
2-3-2-0	0(13.0)	0(10.0)	0(140.0)	LT	4-4-4-1	/(inf)	1(23.0)	1(22.0)	TM !!
2-3-2-1	1(29.0)	1(6.0)	1(4.0)	M	4-4-4-2	/(inf)	/(inf)	1(24.0)	M !!
2-4-0-0	1(11.0)	1(7.0)	1(9.0)	T	4-4-4-3	/(inf)	1(191.0)	1(59.0)	TM !!
2-4-0-2	1(9.0)	1(13.0)	1(8.0)	M	4-4-6-1	/(inf)	1(19.0)	1(15.0)	TM !!
2-4-0-3	1(7.0)	1(9.0)	1(10.0)	L	4-4-6-2	/(inf)	1(232.0)	1(20.0)	TM !!
2-4-2-0	1(9.0)	1(8.0)	1(6.0)	M	4-4-6-3	/(inf)	1(55.0)	1(38.0)	TM !!
2-4-2-1	1(13.0)	1(8.0)	1(9.0)	T	6-2-0-0	0(4.0)	0(3.0)	0(9.0)	LT
2-4-2-3	1(9.0)	1(10.0)	1(8.0)	M	6-2-0-1	0(25.0)	0(23.0)	0(310.0)	LT
2-4-4-0	1(10.0)	1(9.0)	1(7.0)	M	6-2-0-2	0(81.0)	0(76.0)	/(inf)	LT !
2-4-4-1	1(11.0)	1(8.0)	1(9.0)	T	6-2-0-3	1(160.0)	1(141.0)	1(56.0)	M
2-4-4-2	1(9.0)	1(8.0)	1(11.0)	T	6-2-2-1	0(7.0)	0(6.0)	0(24.0)	LT
2-4-6-0	1(9.0)	1(7.0)	1(9.0)	T	6-2-2-2	0(15.0)	0(13.0)	0(90.0)	LT
2-4-6-1	1(407.0)	1(10.0)	1(14.0)	TM	6-2-2-3	0(24.0)	0(22.0)	0(136.0)	LT
2-4-6-3	1(8.0)	1(10.0)	1(7.0)	ML	6-3-0-0	0(33.0)	0(30.0)	0(466.0)	LT
4-2-0-0	0(3.0)	0(3.0)	0(9.0)	LT	6-3-2-0	0(21.0)	0(17.0)	0(199.0)	LT
4-2-0-1	0(13.0)	0(11.0)	0(132.0)	LT	6-4-0-0	1(24.0)	/(inf)	/(inf)	L!!
4-2-0-2	1(23.0)	1(6.0)	1(4.0)	TM	6-4-2-0	1(23.0)	/(inf)	/(inf)	L!!
4-2-0-3	1(43.0)	1(91.0)	1(37.0)	M	6-4-2-1	1(54.0)	/(inf)	/(inf)	L!!
4-2-2-1	0(4.0)	0(3.0)	0(10.0)	LT	8-2-0-0	0(7.0)	0(6.0)	0(19.0)	LT
4-2-2-2	0(7.0)	0(6.0)	0(23.0)	LT	8-2-0-1	0(70.0)	0(55.0)	0(754.0)	LT
4-2-2-3	0(9.0)	0(7.0)	0(33.0)	T	8-2-0-2	0(217.0)	0(170.0)	/(inf)	LT!
4-3-0-0	0(28.0)	0(23.0)	0(631.0)	LT	8-2-0-3	0(430.0)	0(396.0)	/(inf)	LT!
4-3-0-1	1(659.0)	/(inf)	/(inf)	L !	8-2-2-1	0(20.0)	0(18.0)	0(84.0)	LT
4-3-0-2	/(inf)	1(16.0)	1(8.0)	TM	8-2-2-2	0(43.0)	0(38.0)	0(363.0)	LT
4-3-2-0	0(16.0)	0(13.0)	0(248.0)	LT	8-2-2-3	0(76.0)	0(82.0)	0(878.0)	LT
4-3-2-1	/(inf)	0(950.0)	/(inf)	T !!	8-3-0-0	0(50.0)	0(39.0)	0(708.0)	LT
4-3-2-2	1(339.0)	1(10.0)	1(8.0)	TM	8-3-2-0	0(35.0)	0(29.0)	0(282.0)	LT
4-4-0-0	1(14.0)	/(inf)	/(inf)	L !!	8-4-0-0	1(71.0)	/(inf)	/(inf)	L!!
4-4-0-1	/(inf)	1(565.0)	1(585.0)	TM	8-4-2-0	1(85.0)	/(inf)	/(inf)	L!!
4-4-0-2	1(55.0)	1(25.0)	1(16.0)	M	8-4-4-0	1(41.0)	1(18.0)	1(17.0)	M
4-4-2-0	1(16.0)	/(inf)	/(inf)	L !!	8-4-6-0	1(39.0)	1(17.0)	1(16.0)	TM
4-4-2-1	1(137.0)	1(24.0)	1(18.0)	TM					

FIGURE 5.6: Extract of a post-processed logfile - timeout at 1 second

5.5.4 Labeling assessment

The labeling assessment is split into two parts, as already introduced: firstly, the timing performance of the labeling, and secondly, the relevance of the first solutions computed, are assessed. As already introduced in Section 5.4.8 (p70), additional constraints can be applied in order to find the optimal solutions. It is worthy to attest that the implementation of the labeling techniques fulfills the expectation regarding this optimization.

5.5.4.1 Performance analysis

The methodology followed to assess the different labeling techniques (including the variables and values ordering) is similar to what is presented in the previous section.

Using the same systematic tests framework, the tests cases are played using four different labeling techniques, as presented in 5.4.8 (p70):

- the default one,
- the first fail variable ordering,
- the home-made ordering including specific value ordering,
- the reverse home-made ordering. This last option is foreseen to be the worst in terms of performances, representing very little interest, and constitutes more of an academic confirmation.

In order to keep only relevant data, the test cases used during labeling systematic tests are the ones for which the results is known `true`[9]. The test is repeated for the three constraints assignment methods `lineaire`, `tmp` and `memoisation`.

A Python parser similar to the one described in Section 5.5.3 (p74) is written to reshape the logfile from the systematic tests runs into an array humanly easier to read. The files for all the methods are given in Appendix, and an extract is given here under, as example.

```
\%-------------------------------------------------------------------------------------------\%
\%----------------------------AUTOMATICALLY GENERATED RESULT FILE----------------------------\%
\%--------------------------------USING labeling_testbench.py--------------------------------\%
\%-------------------------------------------------------------------------------------------\%
|                    TEST                    |ALL-std[ms]|ALL-ff [ms]| NORM [ms] |  REV [ms] |
|memoisation_steps2_nodes2_edges0_data1.json|    5.0    |    9.0    |    11.0   |    21.0   |
|memoisation_steps2_nodes2_edges0_data2.json|    8.0    |    3.0    |    7.0    |    38.0   |
|memoisation_steps2_nodes2_edges0_data3.json|    7.0    |    3.0    |    6.0    |    24.0   |
|memoisation_steps2_nodes2_edges2_data2.json|    4.0    |    2.0    |    6.0    |    4.0    |
|memoisation_steps2_nodes2_edges2_data3.json|    5.0    |    2.0    |    5.0    |    4.0    |
|memoisation_steps2_nodes3_edges0_data1.json|    14.0   |    6.0    |    22.0   |    61.0   |
|memoisation_steps2_nodes3_edges0_data2.json|    15.0   |    7.0    |    63.0   |   218.0   |
|memoisation_steps2_nodes3_edges0_data3.json|    16.0   |    7.0    |    59.0   |   217.0   |
|memoisation_steps2_nodes3_edges2_data1.json|    10.0   |    5.0    |    11.0   |    12.0   |
|memoisation_steps2_nodes3_edges2_data2.json|    11.0   |    5.0    |    15.0   |   111.0   |
|memoisation_steps2_nodes3_edges2_data3.json|    12.0   |    6.0    |    15.0   |   105.0   |
...
```

Reading the logfile. The reshaped logfile is quite straightforward to read : the first column contains the list of tests that were played, then the four next columns give the time elapsed for the test execution, in ms, for resp. the standard labeling, the first fail variables ordering, the home-made labeling including values and variables ordering using soft constraints, and the *academical* reverse ordering mentioned above.

[9]Typically, the tests taken for the labeling systematic tests are the ones for which the results was `true` in the synchronous systematic tests campaign, as explained in Section 5.5.3 (p74)

5.5.4.2 Tests run

Intuitively, the fastest method is foreseen to be the global first fail, and the slowest the *reverse order home-made* method. The global first fail is known to usually deliver proper results, while the home-made integrates extra computation for reordering and minimizing the values given. The *reverse*, as already expressed, is purely academical and should confirm it is mostly slower than the others. Regarding the standard method, its performance results are quite unpredictable, but are expected to be worst than the first fail results.

Observations.

- As expected, the last column is most of the time much slower than the others, and will not be considered in this section anymore,
- The first column assign a value to the variables as they come, and from the lowest value. The result is not bad – at all –, although the commonly used first fail shows the best results.
- The home-made ordering is slower than the two firsts : this is easily explained by the amount of calculation required to minimize the sum, as explained in Section 5.4.8 (p70).
- Overall, the time taken for the NORM labeling can, in some cases, vary from simple to more than ten times the time taken for the first fail algorithm.
- There are differences in the timing required for the labeling between the three constraints assignment implementations: the `memoisation` method seems better adapted to the home-made ordering.

5.5.4.3 First Solutions analysis

One of the interests of the labeling is to provide *best* solutions first, adding constraints. This was explained in Section 5.4.8 (p70).

Following the different implementations suggested for the labeling, it is possible to compare the first outputs delivered by the system, and verify if it complies with the expected behavior: more relevant solutions out of the home-made variables and values ordering (NORM labeling method).

To emphasize the different labeling behaviors, the selected test input is a pure edges inference: no edge information is given, whilst all the data are given. Indeed this allows a better differentiation between the methods under test as all the variables but `Niveau`'s[10] are free and need to be labeled, while `Niveau` variables are the most constrained ones, hence the less likely to change with labeling methods. The test is performed using the `memoisation` constraints application method – this is arbitrary –, and is repeated for the different labeling techniques.

The graphical representation of the network provided to the system is given in Figure 5.7 (p80), while its JSON corresponding format is given hereunder[11].

```
{"network":[
{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json",
"borneMax":10,
"borneMin":0,
```

[10] and `Delay`, not yet taken into account

[11] At the time of writing these lines, the final JSON file does not yet include all the options, and some are still hardcoded. It concerns the maximum number of solutions to provide, and the sparsity constraint, see Section 5.4.7 (p69)

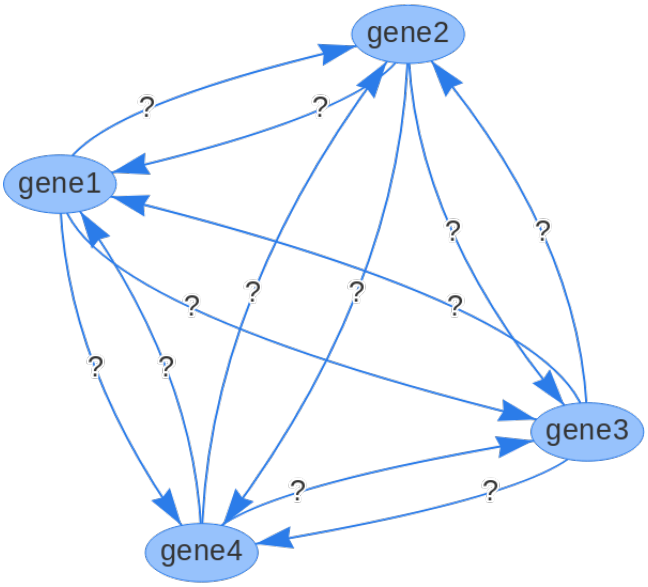

FIGURE 5.7: Graphical representation of the provided network - Labeling assessment


```
"borneEffectOnOthers": 5,
"borneEffectOnSelf":0,
"globalThreshold":9,
"steps":2,
"method":"memoisation",
"nodes":[
{"label":"gene1", "type":"node"},
{"label":"gene2", "type":"node"},
{"label":"gene3", "type":"node"},
{"label":"gene4", "type":"node"}],
"edges": [],
"data": [
{"node":"gene1","step":0,"niveau":5},
{"node":"gene1","step":1,"niveau":6},
{"node":"gene1","step":2,"niveau":7},
{"node":"gene2","step":0,"niveau":0},
{"node":"gene2","step":1,"niveau":2},
{"node":"gene2","step":2,"niveau":4},
{"node":"gene3","step":0,"niveau":0},
{"node":"gene3","step":1,"niveau":1},
{"node":"gene3","step":2,"niveau":2},
{"node":"gene4","step":0,"niveau":9},
{"node":"gene4","step":1,"niveau":7},
{"node":"gene4","step":2,"niveau":5}]}]}
```


The plot of the concentration levels – which are all given in the JSON input file – is in Figure 5.8 (p83). One can recognize the first steps of the ground truth network, as shown in Figure 5.5 (p74).

Results. The ground truth was originally built to have `Delay` variables assigned to 0. Furthermore, as indicated in Section 5.4 (p60), the `memoisation` method does not take the `Delay` into account - those `Delay` variables are not relevant for this labeling methods comparison.
As the concentration levels are provided as inputs, the only information useful to compare the labeling methods in term of solution relevance is the assignment for the interactions variables: `Threshold`, `BorneEffect` and `Effect` variables. The values inferred are summed up in Table 5.1 (p82). The values circled are the changes with respect to the previous solution found.

		Default			First Fail			Home-made		
From	To	Th	BE	Eff	Th	BE	Eff	Th	BE	Eff
First solution										
gene1	gene2	1	2	2	1	2	2	1	5	2
gene2	gene1	1	0	0	1	0	0	1	0	0
gene1	gene3	1	1	1	1	1	1	1	5	1
gene3	gene1	2	0	0	2	0	0	2	0	0
gene2	gene3	1	0	0	1	0	0	1	0	0
gene3	gene2	2	0	0	2	0	0	2	0	0
gene1	gene4	1	2	-2	1	2	-2	1	5	-2
gene4	gene1	1	1	1	1	1	1	1	5	1
gene2	gene4	1	0	0	1	0	0	1	0	0
gene4	gene2	1	0	0	1	0	0	1	0	0
gene3	gene4	2	0	0	2	0	0	2	0	0
gene4	gene3	1	0	0	1	0	0	1	0	0
Second solution										
gene1	gene2	1	(3)	2	1	2	2	1	5	2
gene2	gene1	1	0	0	1	0	0	1	(5)	(1)
gene1	gene3	1	1	1	1	1	1	1	5	1
gene3	gene1	2	0	0	(3)	0	0	2	0	0
gene2	gene3	1	0	0	1	0	0	1	0	0
gene3	gene2	2	0	0	2	0	0	2	0	0
gene1	gene4	1	2	-2	1	2	-2	1	5	-2
gene4	gene1	1	1	1	1	1	1	(8)	5	1
gene2	gene4	1	0	0	1	0	0	1	0	0
gene4	gene2	1	0	0	1	0	0	1	0	0
gene3	gene4	2	0	0	2	0	0	2	0	0
gene4	gene3	1	0	0	1	0	0	1	0	0
Third solution										
gene1	gene2	1	(4)	2	1	2	2	1	5	2
gene2	gene1	1	0	0	1	0	0	1	5	1
gene1	gene3	1	1	1	1	1	1	1	5	1
gene3	gene1	2	0	0	(4)	0	0	2	0	0
gene2	gene3	1	0	0	1	0	0	1	0	0
gene3	gene2	2	0	0	2	0	0	2	0	0
gene1	gene4	1	2	-2	1	2	-2	1	5	-2
gene4	gene1	1	1	1	1	1	1	(9)	5	1
gene2	gene4	1	0	0	1	0	0	1	0	0
gene4	gene2	1	0	0	1	0	0	1	0	0
gene3	gene4	2	0	0	2	0	0	2	0	0
gene4	gene3	1	0	0	1	0	0	1	0	0
Fourth solution										
gene1	gene2	1	(5)	2	1	2	2	1	5	2
gene2	gene1	1	0	0	1	0	0	1	0	(0)
gene1	gene3	1	1	1	1	1	1	1	5	1
gene3	gene1	2	0	0	(5)	0	0	2	0	0
gene2	gene3	1	0	0	1	0	0	1	0	0
gene3	gene2	2	0	0	2	0	0	2	0	0
gene1	gene4	1	2	-2	1	2	-2	1	5	-2
gene4	gene1	1	1	1	1	1	1	(7)	5	1
gene2	gene4	1	0	0	1	0	0	1	0	0

gene4	gene2	1	0	0	1	0	0	1	0	0
gene3	gene4	2	0	0	2	0	0	2	0	0
gene4	gene3	1	0	0	1	0	0	1	0	0
Fifth solution										
gene1	gene2	(2)	2	2	1	2	2	1	5	2
gene2	gene1	1	0	0	1	0	0	1	0	0
gene1	gene3	1	1	1	1	1	1	1	5	1
gene3	gene1	2	0	0	(6)	0	0	(1)	0	0
gene2	gene3	1	0	0	1	0	0	1	0	0
gene3	gene2	2	0	0	2	0	0	2	0	0
gene1	gene4	1	2	-2	1	2	-2	1	5	-2
gene4	gene1	1	1	1	1	1	1	(1)	5	1
gene2	gene4	1	0	0	1	0	0	1	0	0
gene4	gene2	1	0	0	1	0	0	1	0	0
gene3	gene4	2	0	0	2	0	0	2	0	0
gene4	gene3	1	0	0	1	0	0	1	0	0

TABLE 5.1: Labeling assessment - Five first solutions found

Observations. Even on a simple and short example as the JSON file tested, the impact of the labeling method on the solutions relevance is visible. Table 5.1 (p82) allows to compare *intra* solutions and *inter* solutions.

- Considering the first solutions found, the differences are in the values for the `BorneEffect` variables. Indeed, the home-made method automatically assigns the maximum allowed value on the remaining domain. As it's a bound, the value of `BorneEffect` is actually less interesting if not provided by the user as input. The values for the `Effect` variables are alike : it could seem the minimization implemented has no impact. This is due to the limitation of the solutions space, considering the inputs provided.

- The second solution provided reveals how the labeling impacts the search: the "Default" method has the first `BorneEffect` modified, while the "First Fail" has the first `Threshold` modified, only. The difference is related to the remaining domain size when the labeling occurs. While for those two methods, the solutions provided are indeed different in numbers, they represent most likely little interest for the biologist, as they have zero impact on the modeling itself, and cannot really guide the user in its search for the GRN. The home-made method, however, provides a seemingly much more interesting answer : one `Threshold`, one `Effect` and its `BorneEffect` are modified. It is a real other solution.

  This was exactly the purpose of implementing another labeling : obtain most relevant solutions first.

- The next solutions behave the same way : while the two first methods, general and faster (see previous test), provide solutions that did not differentiate much, the home-made labeling allows getting quicker to different possibilities.

5.5.4.4 Conclusion

Labeling is a major aspect of Constraint Logic Programming, and many techniques and heuristics can be set up. Some are more adapted than others when looking at a specific problem, depending in particular on the desired behavior. In this section, it has been shown that the different implementations have strongly different characteristics, and can therefore be used differently, according to the user wish : obtain very quickly a solution, attest that an input

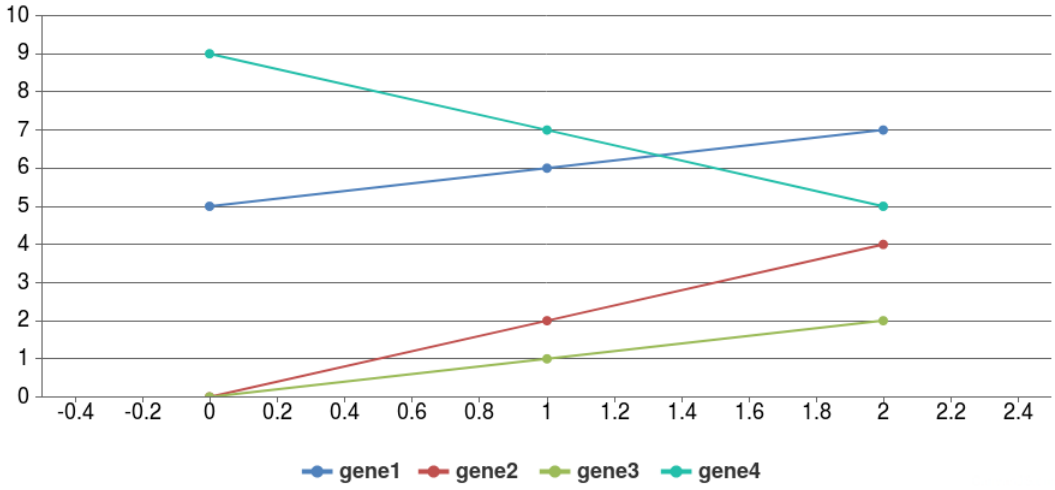

FIGURE 5.8: Concentration levels evolution to infer - Labeling assessment

match a possible network, obtain several relevant solutions, etc. To be thorough, several other labeling techniques can of course be implemented, to better match the users' needs.

5.5.5 Other features tests

This section will go through several main user features that are implemented, and test them from the user perspective. The graphical results come from the Q&D front-end developed for this purpose are detailed in Chapter 6 (p91).

5.5.5.1 Sparsity Constraint

The sparsity constraint can be easily tested, still based on the ground truth file introduced earlier. Using the same test input file as in Section 5.5.4.3 (p79), there are at least solutions with either 4 or 5 interactions for which the `Effect` is different from 0.

Setting the sparsity constraint to have a maximum of 4 interactions produces solutions accordingly, while before, the second solution provided had 5 interactions, as shown in the previous section. The ten first solutions provided are given in Appendix A.6.3 (p150), using JSON representation.

5.5.5.2 Delay Constraint - Synchronous Modeling

The `Delay`, as described in Chapter 3 (p33), can allow the user to explicitly take into account the biological reaction time. The `Delay` for the asynchronous modeling is detailed in a next section.

The usual ground truth test file being built with all the `Delay` variables at 0, another basic network, arbitrary and purely on the purpose of showing the `Delay` effect, is given hereunder in JSON format. Its graphical representation is given in Figure 5.9 (p84). Two genes have an effect on each other, one positive, one negative. The resulting concentration levels graph is given in Figure 5.10 (p84).

```
{"network":
[{
"name":"delay",
"borneMax":5,
"borneMin":0,
"borneEffectOnOthers":3,
"borneEffectOnSelf":0,
```

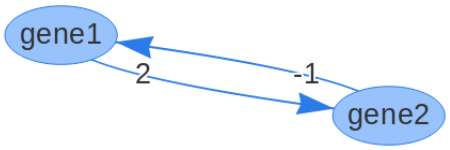

FIGURE 5.9: Graphical representation of the network - Delay case


```
"globalThreshold":1,
"steps":5,
"method":"lineaire",
"nodes":[
{"label":"gene1", "type":"node"},
{"label":"gene2", "type":"node"},
],
"edges": [
{"id":1,"from":"gene1","to":"gene2","threshold":1,"borneEffect":3,"delay":0,"effect":2},
{"id":2,"from":"gene2","to":"gene1","threshold":1,"borneEffect":3,"delay":0,"effect":-1},
],
"data": [
{"node":"gene1", "step":0, "niveau":5},
{"node":"gene2", "step":0, "niveau":0}
]}]}
```


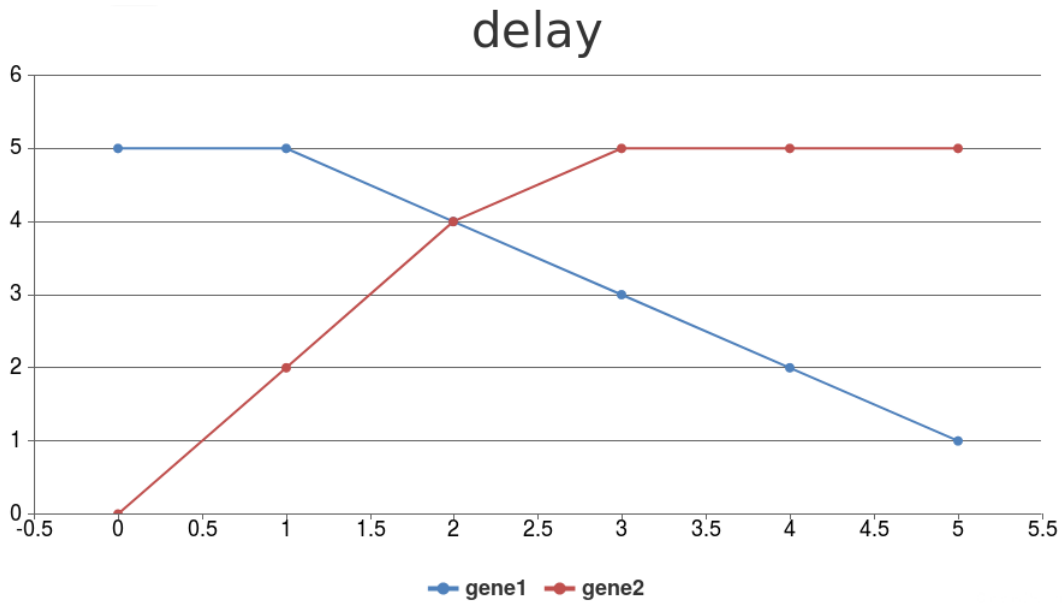

FIGURE 5.10: Concentration levels evolution - Basic case

Delay introduced as input. The tool offers the opportunity, for the synchronous modeling using either `lineaire` or `tmp`, to add a `Delay`, as explained in Section 3.4 (p36).

The input test file can be modified in order to include a delay in the interaction $gene2 \rightarrow gene1$, for example of 1 time step. The intuition is to observe the effect of this interaction 1 time step later in the resulting concentration levels inference.


```
...
"edges": [
{"id":1,"from":"gene1","to":"gene2","threshold":1,"borneEffect":3,"delay":0,"effect":2},
{"id":2,"from":"gene2","to":"gene1","threshold":1,"borneEffect":3,"delay":1,"effect":-1},
],...
```

The system computes the concentration levels knowing this `Delay` introduced. The resulting concentration levels are graphically shown in Figure 5.11 (p85). As expected, the output shows that the evolution of the concentration levels of *gene*1 are delayed by one time step : the interaction becomes effective only after a delay, when the `Threshold` is crossed.

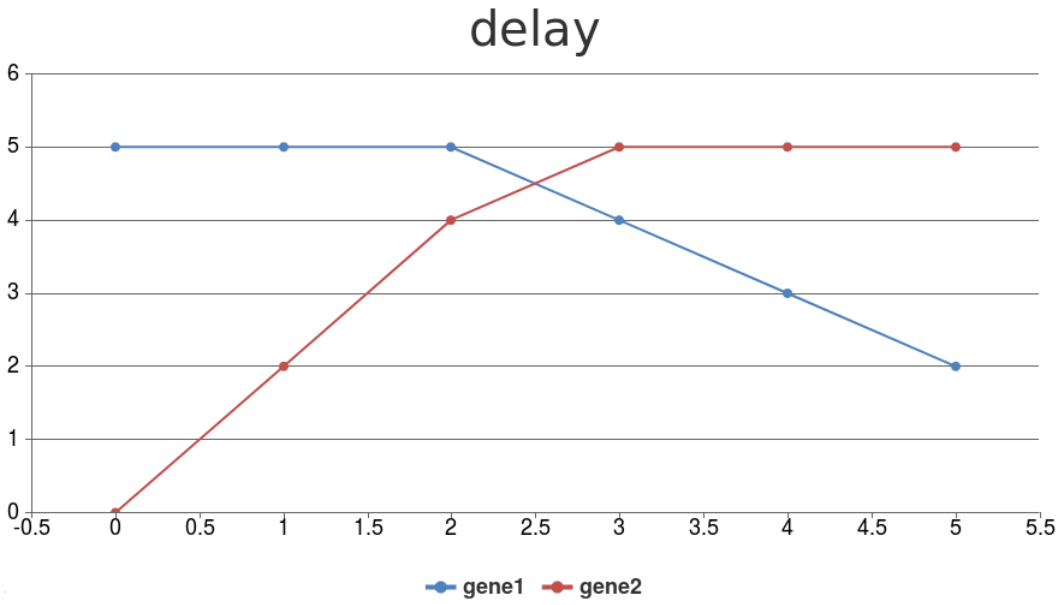

FIGURE 5.11: Concentration levels evolution - Delay case

Delay inferred. Currently, the `Delay` variables inference is not available for the synchronous methods.

5.5.5.3 Control entity

Up to now, the accent has been put on the "node" type entities, which are basically part of the whole regulatory process. The entity type "control" has been introduced to handle situations where a substance is purely under control, that is, where one can control externally fully the concentration levels, while still this substance can have an impact on the others.

An example of use is given here under, for the tiny network represented in Figure 5.12 (p86). As the goal is to enlighten the use of the `control` substance, only two genes participate to the system, and the interactions are forced to 0. Yet, the data given as inputs are such as there is a need for a regulatory mechanism, as the concentration levels change over time. A `control` substance is added, in order to take in charge those interactions, and it is supposed that the concentration levels of this substance are given for each time steps.

```
{"network":
[{
"name":"control_substance",
"borneMax":10,
"borneMin":0,
"borneEffectOnOthers":3,
"borneEffectOnSelf":0,
"globalThreshold":1,
"steps":5,
"method":"tmp",
"nodes":[
{"label":"gene1", "type":"node"},
{"label":"gene2", "type":"node"},
{"label":"substA", "type":"control"},
],
```

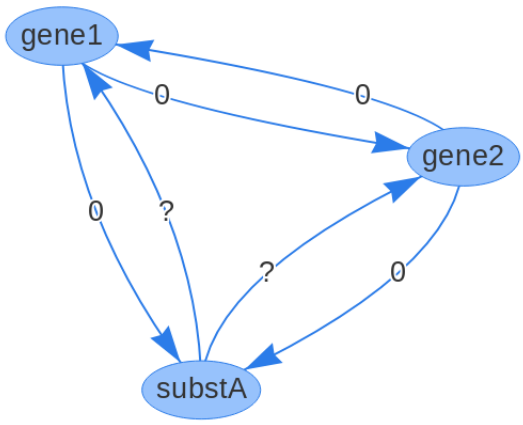

FIGURE 5.12: Representation of the network - Control entity


```
"edges": [
{"id":1,"from":"gene1","to":"gene2","threshold":"/","borneEffect":"/","delay":0,"effect":0},
{"id":2,"from":"gene2","to":"gene1","threshold":"/","borneEffect":"/","delay":0,"effect":0}
],
"data": [
{"node":"gene1", "step":0, "niveau":0},
{"node":"gene1", "step":5, "niveau":5},
{"node":"gene2", "step":0, "niveau":9},
{"node":"gene2", "step":5, "niveau":0},

{"node":"substA", "step":0, "niveau":6},
{"node":"substA", "step":1, "niveau":8},
{"node":"substA", "step":2, "niveau":3},
{"node":"substA", "step":3, "niveau":7},
{"node":"substA", "step":4, "niveau":9},
{"node":"substA", "step":5, "niveau":2}
]}]}
```


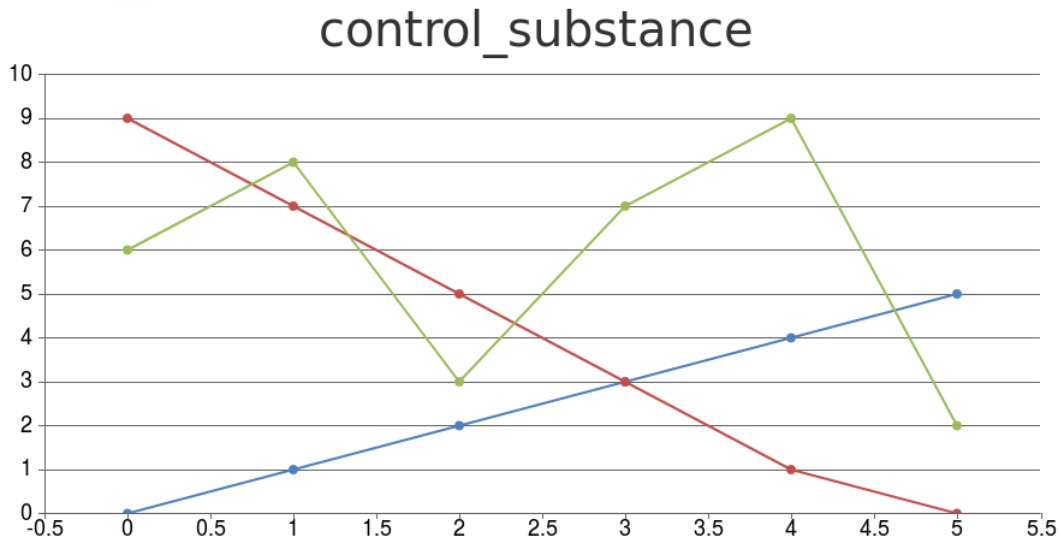

FIGURE 5.13: Concentration levels evolution with a substance of type `control`

The graphical results of the inference is given in the Figure 5.13 (p86) and the interactions of the network resulting in those concentration levels are given in the JSON output extract, below. Two interactions have been inferred having an `Effect` different from 0. As expected, the `control` substance fully played its role : interacting with the others, while having a fully externally controlled concentration level.

```
"edges":[
{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene2","to":"gene1","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"substA","threshold":0,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"substA","to":"gene1","threshold":1,"borneEffect":3,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"substA","threshold":0,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"substA","to":"gene2","threshold":1,"borneEffect":3,"delay":0,"effect":-2}]
```


5.5.5.4 Asynchronism Modeling

The asynchronism modeling has been largely discussed in Chapter 3 (p33).

As a reminder, four assumptions were made prior to the implementation. Those assumptions are written in Section 3.4.1.1 (p36). As a quick reminder, the asynchronous case refers to single change per time step : only one gene will see its expression level being modified per time step.

This of course has an impact on the data to provide : two genes shall not experience a change at the same time step in the inputs given to the system, or no solution will be found. Similarly, when providing interactions, no two edges shall have the same `Delay` entered, as this variable specifically tells when the change occurs: the same `Delay` twice would mean two changes at the same time step, which is incorrect in this modeling.

According to those considerations, the usual ground truth test file cannot be used. A dedicated network, arbitrarily tiny for the need of this section, has been created.

First, the inference of the interactions based on a subset of data is shown. Then, the input file will be modified to leave only the interactions between genes and the initial conditions, to attest the system can compute the concentration levels based on the interactions provided.

Interactions inference. The input file is given hereunder: are provided the entities – two genes – and several concentration levels. Those are selected to arbitrarily guide the behavior of the inference: there should be a positive effect on *gene*2 and a negative effect on *gene*1.


```
{"network":
[{
"name":"asynchronous_test",
"borneMax":5,
"borneMin":0,
"borneEffectOnOthers":3,
"borneEffectOnSelf":0,
"globalThreshold":1,
"steps":10,
"method":"asynchronous",
"nodes":[
{"label":"gene1", "type":"node"},
{"label":"gene2", "type":"node"}
],
"edges": [],
"data": [
{"node":"gene1","step":3,"niveau":3},
{"node":"gene1","step":0,"niveau":5},
{"node":"gene2","step":1,"niveau":2},
{"node":"gene2","step":0,"niveau":1}]}]}
```


The results of the concentration levels is given in Figure 5.14 (p88), and the complete JSON file, including the edges inferred, is given hereunder. The result matches the expectation, with the foreboded `Effect` values assigned. Interesting to note, starting from step 6, the interaction $gene1 \rightarrow gene2$ has no effect anymore, as the value of *gene*1 is below the activation threshold. *gene*2 is therefore *stuck* at a concentration level of 4, instead of the upper bound, 5.


```
{"network":[{
"name":"asynchronous_test",
"borneMax":5,"borneMin":0,"borneEffectOnOthers":3,"borneEffectOnSelf":0,
"globalThreshold":1,"steps":10,"method":"asynchronous",
"nodes":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"}],
"edges":[
```

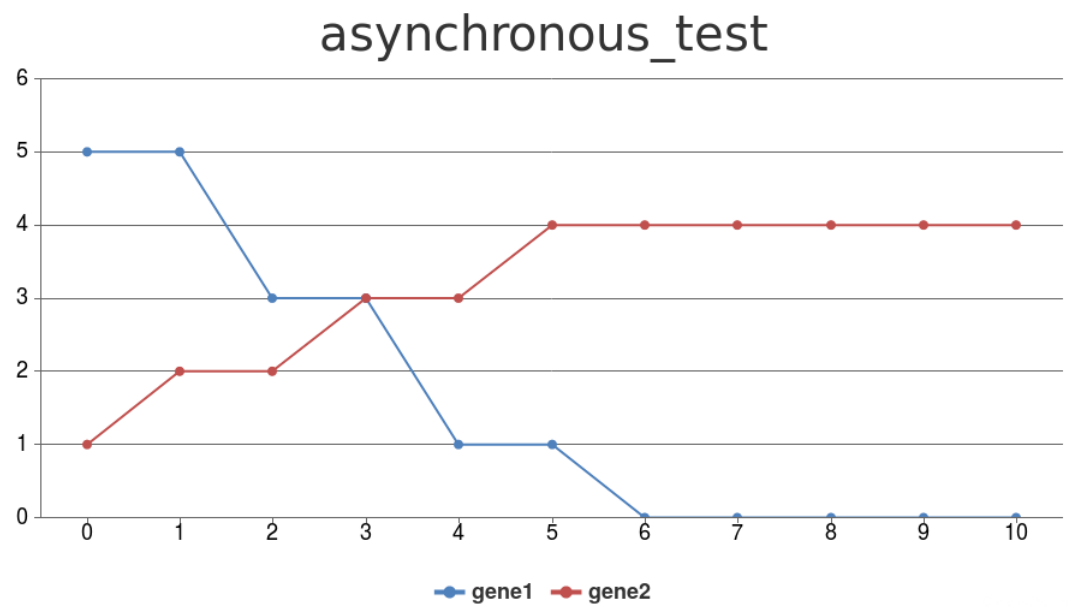

FIGURE 5.14: Concentration levels evolution using `asynchronous` modeling


```
{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":3,"delay":1,"effect":1},
{"id":"toChange","from":"gene2","to":"gene1","threshold":1,"borneEffect":3,"delay":0,"effect":-2}],
"data":[
{"node":"gene2","step":10,"niveau":4},
{"node":"gene2","step":9,"niveau":4},
{"node":"gene2","step":8,"niveau":4},
{"node":"gene2","step":7,"niveau":4},
{"node":"gene2","step":6,"niveau":4},
{"node":"gene2","step":5,"niveau":4},
{"node":"gene2","step":4,"niveau":3},
{"node":"gene2","step":3,"niveau":3},
{"node":"gene2","step":2,"niveau":2},
{"node":"gene2","step":1,"niveau":2},
{"node":"gene2","step":0,"niveau":1},
{"node":"gene1","step":10,"niveau":0},
{"node":"gene1","step":9,"niveau":0},
{"node":"gene1","step":8,"niveau":0},
{"node":"gene1","step":7,"niveau":0},
{"node":"gene1","step":6,"niveau":0},
{"node":"gene1","step":5,"niveau":1},
{"node":"gene1","step":4,"niveau":1},
{"node":"gene1","step":3,"niveau":3},
{"node":"gene1","step":2,"niveau":3},
{"node":"gene1","step":1,"niveau":5},
{"node":"gene1","step":0,"niveau":5}]}]}
```


Concentration levels inference. The edges inferred during the first test can be used as inputs for the test in the other way around : giving the edges and inferring the complete concentration levels flow. It is important to note that, in order to avoid the trivial case where the concentration levels are zero all along, initial conditions are set. They slightly differ from the previous test file, as *gene*2 starts at a level of 2. The input file becomes:


```
{"network":
[{
"name":"asynchronous_test",
"borneMax":5,
"borneMin":0,
"borneEffectOnOthers":3,
"borneEffectOnSelf":0,
"globalThreshold":1,
"steps":10,
"method":"asynchronous",
"nodes":[
{"label":"gene1", "type":"node"},
```

```
{"label":"gene2", "type":"node"}
],
"edges": [
{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":3,"delay":1,"effect":1},
{"id":"toChange","from":"gene2","to":"gene1","threshold":1,"borneEffect":3,"delay":0,"effect":-2}],
"data": [
{"node":"gene2","step":0,"niveau":2},
{"node":"gene1","step":0,"niveau":5}
]}]}
```


The results are quite convincing, as the flow is easily inferred. Results are graphically shown in Figure 5.15 (p89)

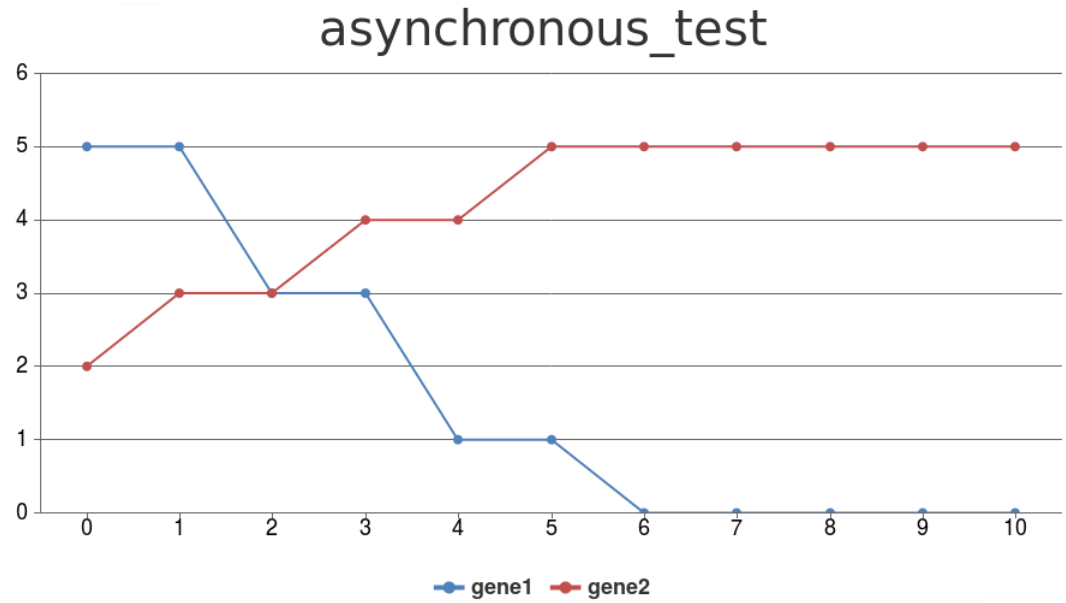

FIGURE 5.15: Concentration levels evolution using `asynchronous` modeling, input: edges

5.6 Conclusion

This chapter covered the implementation of the back-end of the tool developed.

Firstly, the code structure is introduced and the layered view detailed, setting in perspective the development of the REST API, and the core of the service offered: the GRN inference tool. The concrete implementation of the Constraint Logic Program of the tool is presented, from the user input as JSON data in a HTTP request, up to the output JSON data in the HTTP response. In particular, the implementation of the different steps of a CLP are detailed: association list, modeling and labeling. As the heuristic development and optimization is an important part of the CLP paradigm, a systematic tests framework is developed and detailed. This framework allows to differentiate implementations methods used in the modeling and labeling part. The methodology to use this systematic tests framework is specified, an intuition of the expected results is provided, followed by the actual tests runs observations and conclusions. The base for heuristics development is set in place.

Several tests on alternate development such as the sparsity constraints and `Delay` inferences or `asynchronous` modeling method are also achieved, to verify the high level behavior of the implemented features.

Overall, this chapter constitutes the core of this thesis, including the CLP development and assessment.

Chapter 6

Front-End development

6.1 Introduction

The need of a user interface has already been discussed in Chapter 4 (p41). While the back-end (see Chapter 5 (p57)) can be accessed with computer-expert software such as `cURL`[1], Lucy, introduced in Chapter 4 (p41), prefers, for obvious reasons, working with a dedicated front-end. As discussed in Section 4.4.1 (p47), this front-end uses plain HTML/CSS/JavaScript technologies, and is limited in terms of functionalities. The purpose of this front-end is to have a concrete implementation that can be used during development, and help suggest new development ideas, rather than a fully developed and production-grade user interface[2].

This chapter details how the front-end is implemented and can be used.

6.2 Overview of the front-end

As introduced in Section 4.4.1 (p47), the front-end is built as a single page application. Lucy stays on a single Web page, which content is refreshed according to her actions. It allows a more dynamic behavior as there is no need to refresh the complete page, but only the modified element.

The requirements to fulfill are elicited in Section 4.2 (p41), and reminded below in the form of user stories. Considering the context, there is no specific needs to express these user stories in more formal requirements.

- As a user, I want to provide the entities, part of the system I want to study with the tool.
- As a user, I want to provide data so that the GRN can be inferred.
- As a user, I want to provide the interaction rules between genes so that data expressing the concentration levels of the gene products can be generated.
- As a user, I want to be able to manually modify the interaction rules suggested by the tool.
- As a user, I want to be able to visualize the results of the inference or data generation.

To implement these functionalities in a structured way, a solution based on tabs is developed: a first tab is dedicated to the inputs, a second tab is dedicated to the outputs and post-treatment, and a third tab currently provides help and indications.

[1] **https://curl.haxx.se/docs/manpage.html**.

[2] The separation of concerns principle allows as many front-end development as necessary to become, later, production-ready

6.3 Inputs tab

This tab allows Lucy:

- to provide her inputs regarding the network to study,
- to send the request to the server.

Section 4.5 (p51) introduced the DSL used in the tool to define a gene regulatory network, and to provide the back-end with all required information to infer the network. These inputs are formatted into a JSON file sent to the back-end. Following the guideline driving this front-end development, an HTML text area environment is provided to write this JSON input file, directly. This is the main element of the Inputs tab, as visible in Figure 6.1 (p92). The JSON input file must comply with the specifications given in Section 4.5 (p51). In order to provide some assistance, a help tab gives a complete exemple of a correct JSON input.

When Lucy has written a correct JSON input file, she presses on the **Process** button, see Figure 6.1 (p92). This will compile the information and properly set the request to the server.

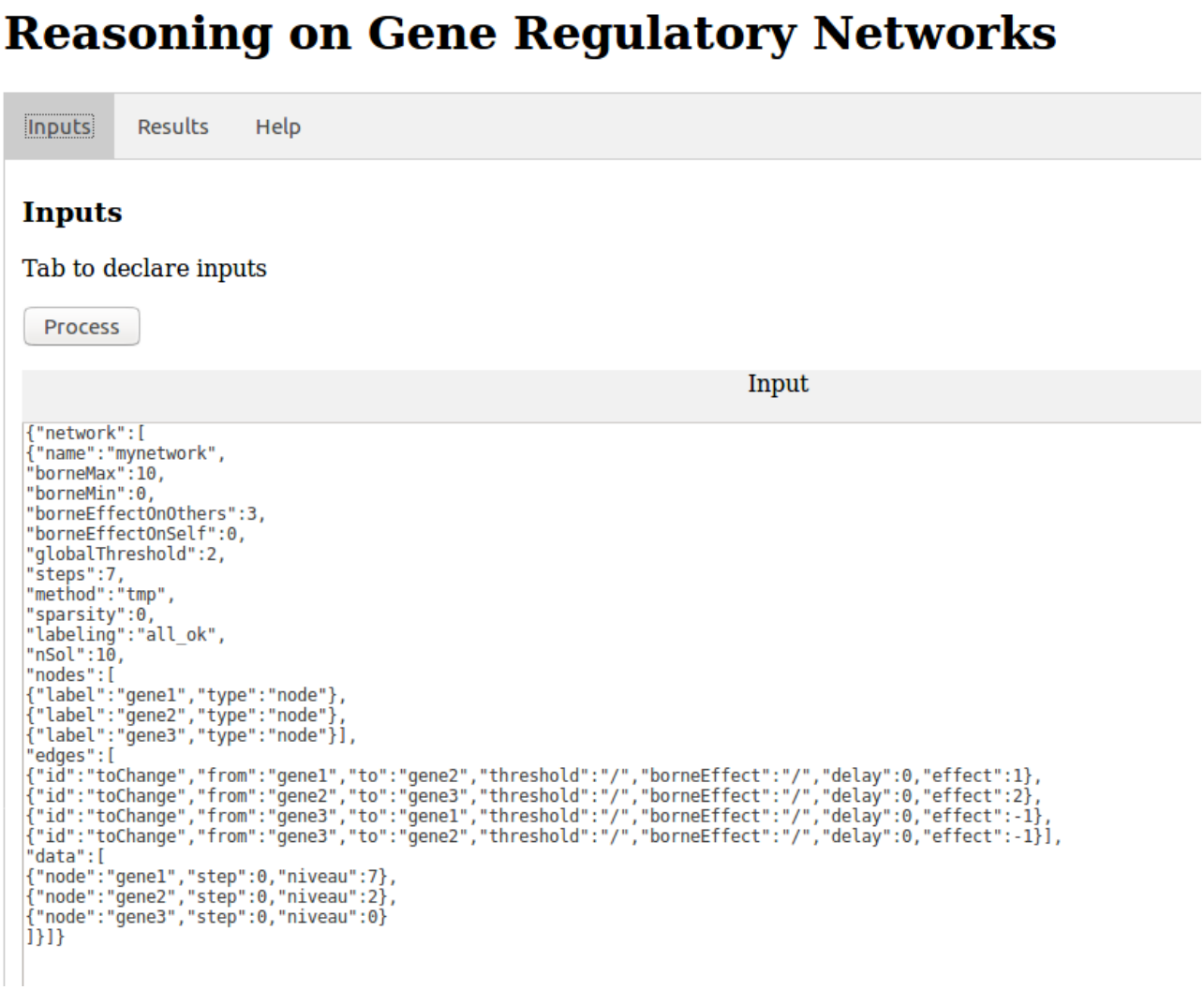

FIGURE 6.1: Inputs tab

This tab implements the four first requirements regarding the front-end in a straightforward way. Improvements can be foreseen as discussed in Chapter 8 (p105).

6.4 Results tab

The Results tab provides the results of the inference realized by the back-end. In the nominal case, the server computes the results and compiles them according to the specified JSON file, as per Section 5.4.1 (p60). The response having a HTTP Status 201 is processed by the front-end to properly display the results to Lucy.

The results are provided in two formats: a graphical view of the solutions, and the JSON output file computed. An illustration of the Results tab is given in Figure 6.2 (p93). Although the processing of the results are limited with this first version of the front-end, the most important and required functionalities are available.

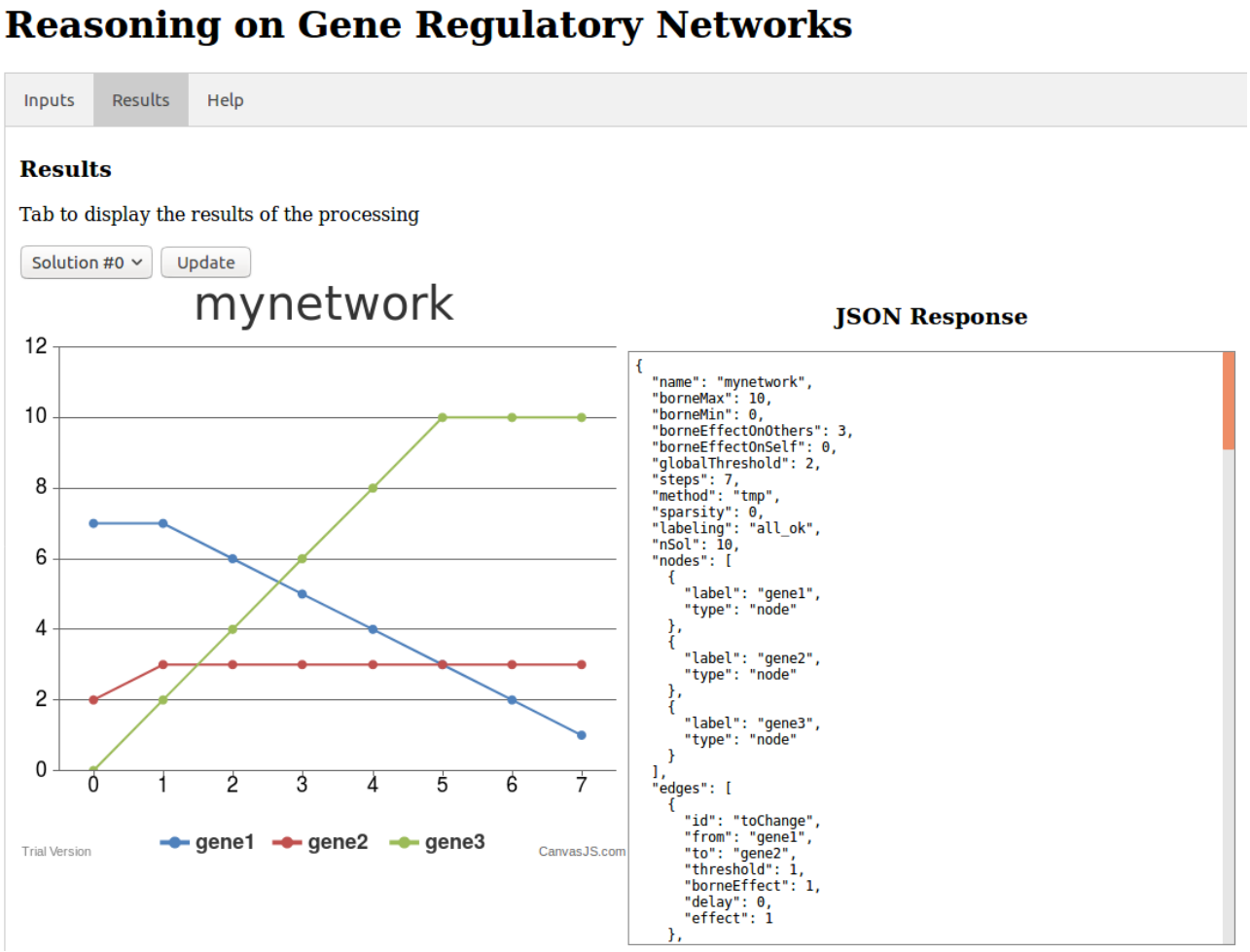

FIGURE 6.2: Results tab

6.4.1 Graphical view of the solutions

As shown in Figure 6.2 (p93), the left part of the screen offers a graphical view of the solution. This graph is automatically refreshed as soon as a new JSON file is received from the server, or when another solution is selected by the user, as explained in Section 6.4.2 (p94).

Based on the JSON response data, the front-end code extracts the name of the network, its entities and their corresponding concentration levels for all time steps.

The graphical layout is implemented using a dedicated JavaScript library, **canvas.js**, see [5]. This library has been chosen as it is particularly adapted to the desired functionality, for

its ease of use and its extensive documentation. **canvas** generates the graph based on the data given, which are directly extracted from the JSON response file received from the server.

6.4.2 Multiple Solutions

In the JSON input file, Lucy can specify, with the parameter `nSol`, the total number of solutions she would like the back-end to compute[3]. After the response is received from the server, the interest for Lucy is to be able to visualize all those solutions.

A possibility would be to open new HTML pages containing the results for all computed solutions. This is however not acceptable, as, theoretically, in case of under-constrained networks, the number of solutions can be quite high. Limiting this number because of the results rendering is not appropriate. The front-end must be adapted to manage a large number of solutions received from the back-end.

Another possibility is to simply provide a drop-down list containing a reference to each solutions. Lucy, after selecting the desired solution to show, clicks on the **Update** button provided, which refreshes the page accordingly. This solution has been implemented.

Before Lucy has selected a specific solution to be rendered, the very first solution computed is displayed by default, and the global response JSON file is provided. When Lucy then selects the solution she wants to analyze and clicks on the **Update** button, the graph is updated according to the selected solution's data, and only the corresponding JSON file part is shown.

A print screen of this functionality is given in Figure 6.3 (p95). The drop-down list allows selecting the desired solution, and the **Update** button allows refreshing the page accordingly.

The drop-down list is refreshed every time a new response is received from the server: the items in the list are removed, and new ones are added on the fly. This is yet another example of the use of the Ajax techniques that were introduced in Section 4.4.1 (p47).

The detail of the code is found in Appendix A.7 (p161).

[3]This is introduced in Section 4.5 (p51)

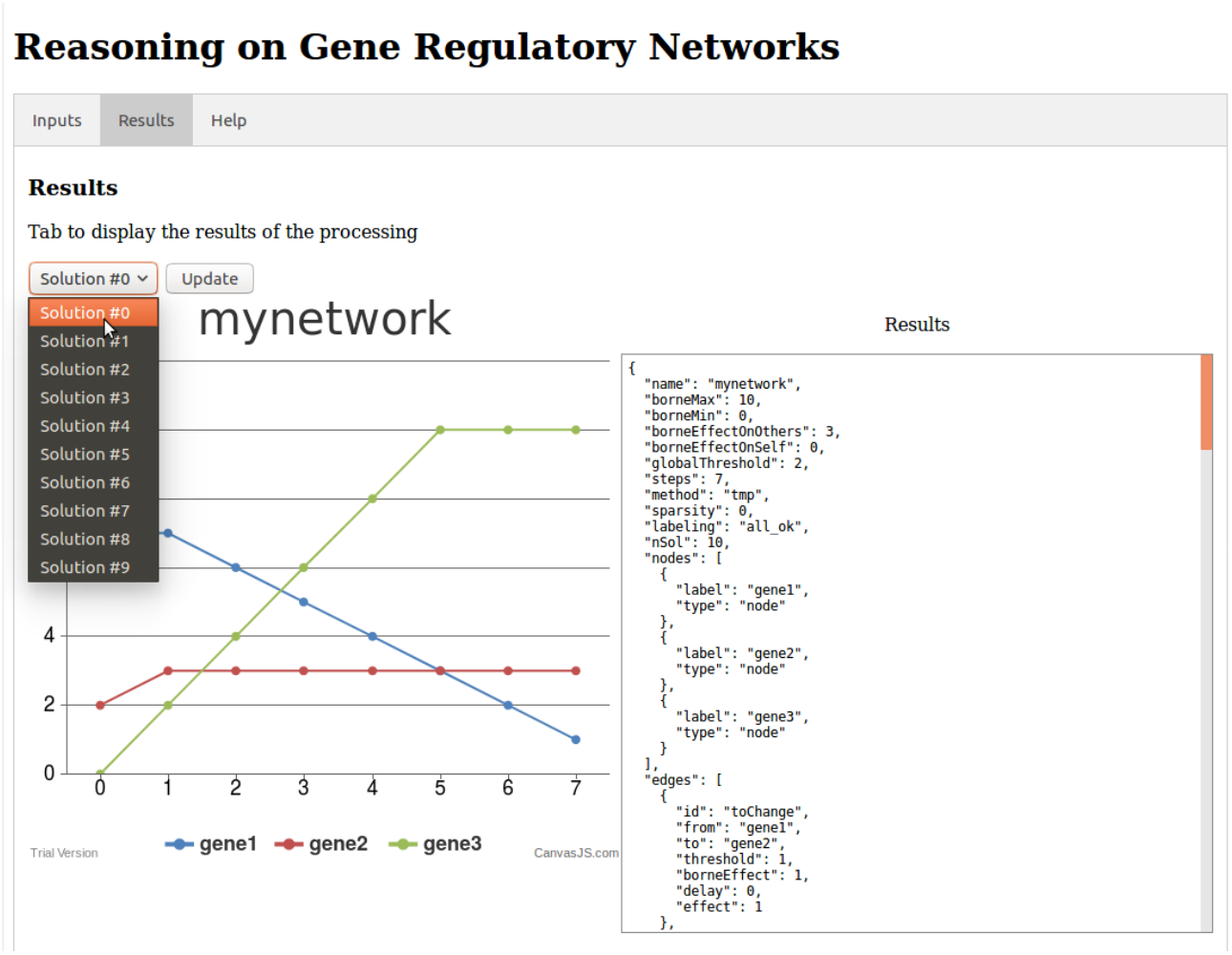

FIGURE 6.3: Visualization of the results computed - multiple solutions provided

Chapter 7

Case-based application of the tool

7.1 Introduction

The tool has been detailed from a general perspective in Chapter 4 (p41), from a back-end perspective in Chapter 5 (p57) and from a user perspective in Chapter 6 (p91).

This chapter shows how the concrete implementation is used, based on a more realistic case: the lac operon. This regulatory mechanism is introduced and explained in Section 1.4 (p14). Different use cases are shown: when the interactions and initial conditions are known, and when the experiment data are available – after normalization and noise reduction. These situations match precisely the requirements as elicited in Section 4.2 (p41).

The goal of this chapter is not to thoroughly analyze the well-known lac operon GRN, but to show how Lucy can use the tool developed on a practical cases.

7.2 Network assessed

The first step when assessing gene regulatory networks using the tool developed is to define the network of interest. Considering the use case, several informations are required. For instance, if Lucy wants to infer the interactions based on experiment data, at least several data points must be provided.

In order to facilitate the understanding of the use cases discussed in this chapter, a simplified modeling of the lac operon system is used. Considering the results of the transcription in Table 1.2 (p17), five entities – nodes in the tool terminology – are part of the system: glucose, lactose, CAP-cAMP, lacI and operonAYZ.

The global network and inference parameters are defined arbitrarily. As per the user input JSON file, those parameters are given in Table 7.1 (p98).

7.3 Case 1: Interactions and initial conditions

For this first use case, although the lac operon system is quite known, Lucy wishes to assess some parameters of the interactions. She desires to provide to the tool only the interactions – the edges of the network – and let the tool rebuild all the data.

The network to study is given in Figure 7.1 (p98). The content of the corresponding JSON file is given here under.

In order to avoid the trivial solution[1], Lucy sets some initial conditions as defined in Table 7.2 (p99). Those conditions relate the state in which the system is at the very beginning of the

[1] All concentration levels equals to 0, as if no entity were in the system under test

Parameter	Value
name	lac operon
borneMax	10
borneMin	0
borneEffectOnOthers	3
borneEffectOnSelf	0
globalThreshold	2
method	tmp
sparsity	0
labeling	all_ff
nSol	10

TABLE 7.1: Global parameters used in the context of the detailed cases

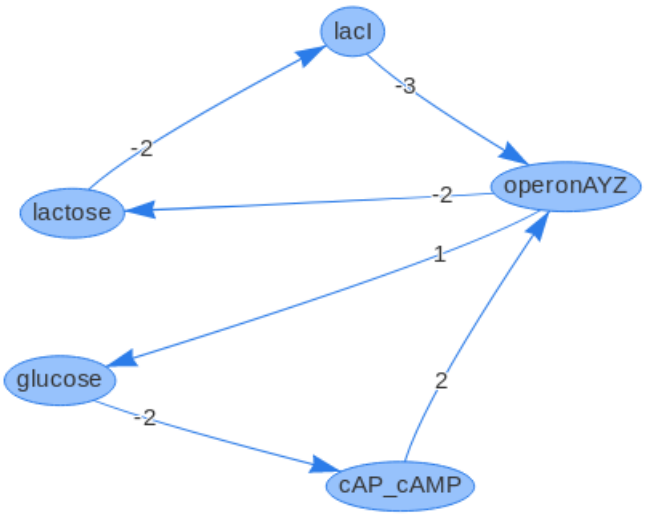

FIGURE 7.1: Lac operon network assessed - case 1

experiment. Lucy sets those initial conditions corresponding to a state where there is a lot of lactose, as well as operonAYZ in the system. Lucy selects the other concentration levels to be compatible with previous experiments realized.

Lucy expects the concentration to be modified according to Table 1.2 (p17): the lactose concentration decreasing, and the glucose concentration increasing. The operon concentration level should also rise, up to an equilibrium, and then decrease as the glucose concentration would be high enough to limit the CAP-cAMP effect, while the lacI effect would rise.

7.3.1 First steps

As a good modeling practice, the first trial is performed with a small number of steps. Using the tool on the network, the results are given within a few seconds, as per Figure 7.2 (p99). As expected, the lactose concentration level decreases, and the glucose concentration level

Entity	Concentration level
lactose	9
lacI	2
operon	7
cAP-cAMP	8
glucose	1

TABLE 7.2: Initial conditions - case 1

increases. During these steps, the operon concentration level reaches the maximum value defined as input.

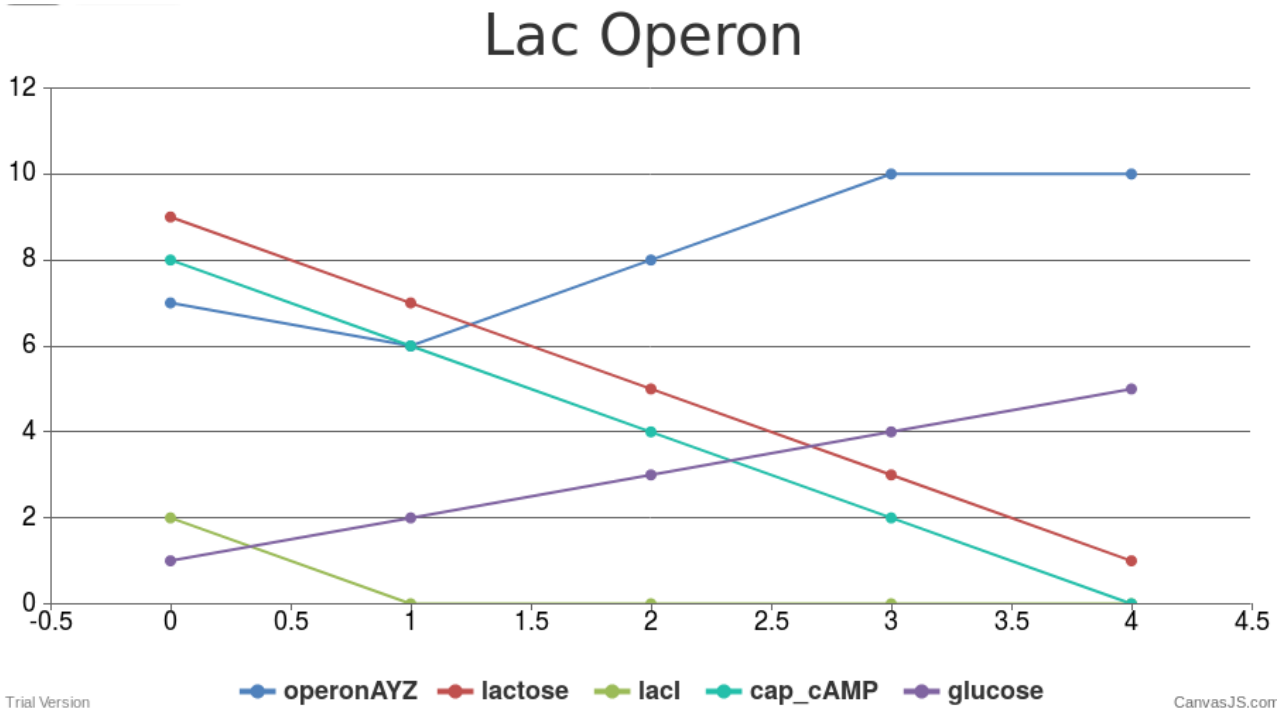

FIGURE 7.2: Lac Operon network assessed

7.3.2 Equilibrium

From a modeling perspective, an important element is to be observed on the first steps results in Figure 7.2 (p99): the equilibrium is expected to happen thanks to the glucose limiting the production of CAP-cAMP, and the lacI being produced in the absence of lactose. The first effect is well modeled: the concentration level of CAP-cAMP decreases as the glucose concentration rises in the first steps. The second effect, linked to the absence of lactose, is not modeled properly: the lacI concentration level, that is supposed to increase while the lactose concentration decreases, stays at its minimum level even when lactose concentration level decreases.

This is easily explained by the fact the modeling does not integrate the complete system and all the interactions, but only a subset.

The tool allows to correct this modeling to make it consistent with the experiment: Lucy can add a `control` entity, as introduced in Section 4.5 (p51). The concentration levels of such entity are all imposed (and not subject to interactions from other entities) while interactions can be forced. Lucy decides, to model the continuous production of lacI, to add a `supportLacI` entity, that has a positive effect on the concentration level of the lacI. Lucy fixes the `Effect` of the interaction at 1. The updated network is given in Figure 7.3 (p100)

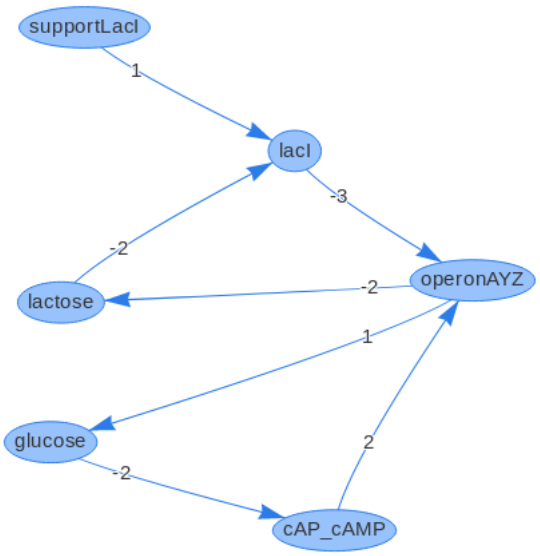

FIGURE 7.3: Lac operon network assessed updated with `control` entity

As the network is fixed, two good practices should be added to speed up the processing:

1. Applying a sparsity constraint, that will limit the engine in its research for other interactions,
2. Setting all the `Effect` not mentioned in the network of Figure 7.3 (p100) to 0, which will drastically help pruning the search tree, reinforcing the effect of the sparsity constraint.

The final JSON input file is given in Appendix A.6.2 (p149). This file can be copy-pasted in the text area of the Inputs tab in order to run the complete experiment. The print screen of the front-end with the JSON input file is given in Figure 7.4 (p101).

The results obtained from the back-end are shown in Figures 7.5 (p102) and 7.6 (p103). The concentration levels follow the expected behavior:

- the lactose concentration level decreases and, as there is no other lactose input, it stays low,
- the glucose concentration level increases as the concentration level of the operon allows its synthesis,
- as the glucose concentration level increases, the CAP_cAMP concentration level decreases,
- the control substance now allows the lacI to be produced when there is no lactose,
- the operon concentration level start decreasing when the lacI concentration is high enough.

Confident that this base matches the reality, Lucy can now modify the parameters[2] one by one, and run again the experiment to compare the results and make a potential biological breakthrough, thanks to the tool developed.

[2]The `Threshold`'s, the `Delay`'s, the `Effect`'s, for instance

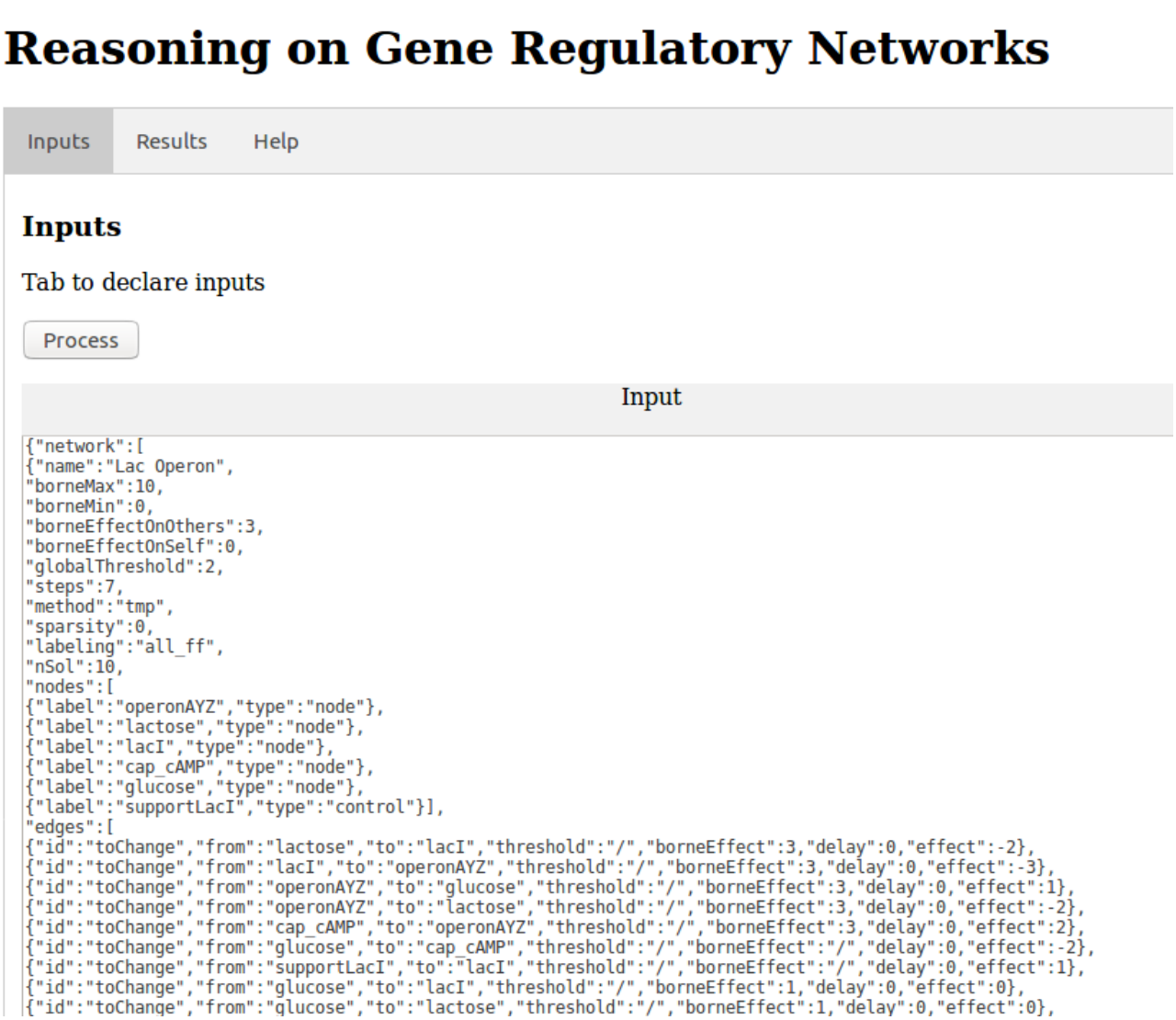

FIGURE 7.4: Print screen of the Inputs tab of the complete experiment

7.4 Case 2: Inferring based on experimental data

In the previous use case, the edges were known and the data unknown, despite some initial conditions. The tool can be used the other way around: Lucy can provide the entities and the data only, without any edges, and the tool can infer the interactions between the entities that can explain, considering the modeling chosen (see Section 5.4.6 (p65)), the experimental data.

In the context of this thesis, this can be easily done starting from the results of the previous use case. The input file is the JSON response from the server, with the edges removed from the file. This is a simple way of getting rid of the normalization and filtering of experimental data. The general parameters can be left unchanged.

Figure 7.7 (p104) shows the Inputs tab, with no edge provided to the system as inputs. When clicking on the **Process** button and sending the request, the back-end will infer the interactions, and send back a JSON response file containing the definition of those interactions. It is expected to retrieve the interactions from the original JSON input file, or – which would even be better in terms of relevance of the inference – suggest other interactions that can explain the experimental data.

The front-end could be improved for this situation in order to show a network graph of the inferred interaction. This is discussed in Chapter 8 (p105).

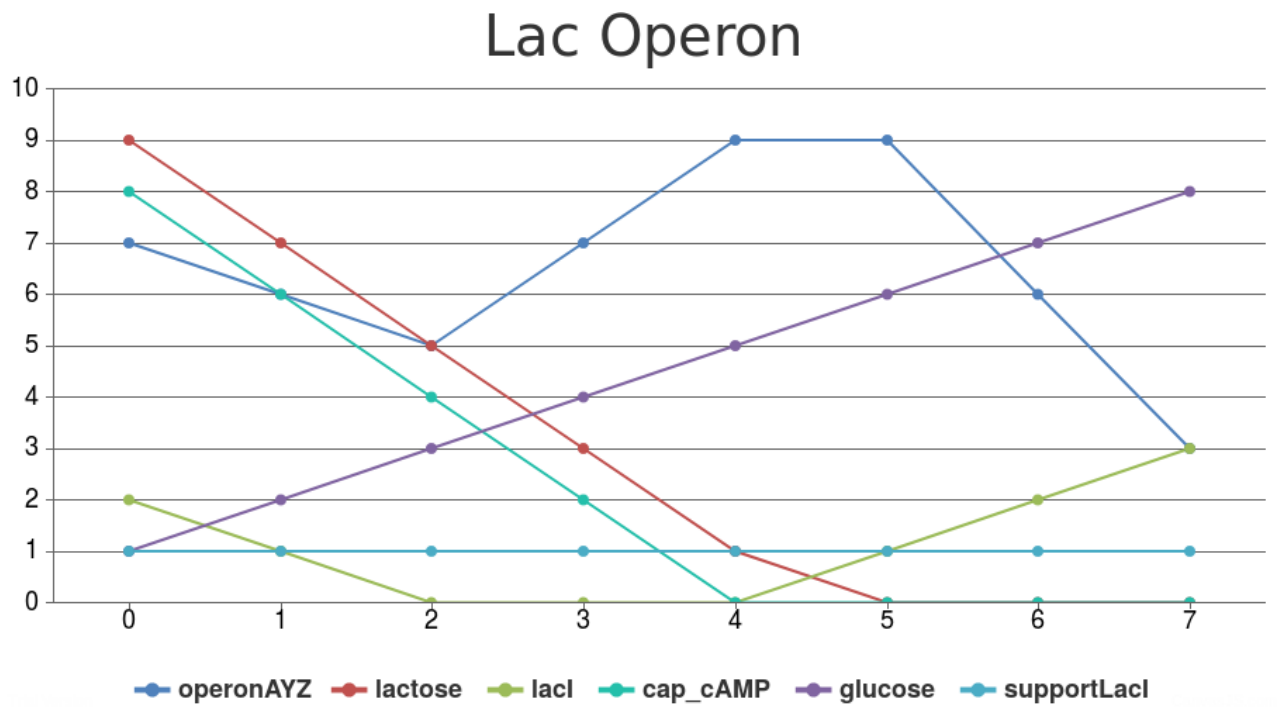

FIGURE 7.5: Result of the complete experiment, with the network given in Figure 7.3 (p100)

The inferred interactions are given here under[3], and provided as a graph in Figure 7.8 (p104). The results are beyond expectations: within a few seconds, the interactions are computed, and are, for some, different from the input file of the use case developed in Section 7.3 (p97).

```
...
{"id": "toChange","from": "operonAYZ","to": "lactose","threshold": 1,"borneEffect": 2,"delay": 0,"effect": -2},
{"id": "toChange","from": "operonAYZ","to": "lacI","threshold": 1,"borneEffect": 1,"delay": 0,"effect": 1},
{"id": "toChange","from": "lacI","to": "operonAYZ","threshold": 1,"borneEffect": 3,"delay": 0,"effect": -3},
{"id": "toChange","from": "operonAYZ","to": "cap_cAMP","threshold": 1,"borneEffect": 2,"delay": 0,"effect": -2},
{"id": "toChange","from": "cap_cAMP","to": "operonAYZ","threshold": 1,"borneEffect": 2,"delay": 0,"effect": 2},
{"id": "toChange","from": "cap_cAMP","to": "lacI","threshold": 1,"borneEffect": 2,"delay": 0,"effect": -2},
{"id": "toChange","from": "operonAYZ","to": "glucose","threshold": 1,"borneEffect": 1, "delay": 0,"effect": 1},
...
```

From there onwards, Lucy can start her biological work, assessing the results based on her expertise, and iterate with the tool using other inputs.

7.5 Conclusion

This chapter detailed two use cases where the tool helped Lucy, the biologist, in her reasoning over the network of the lac operon. It has shown the steps and the methodology to use the tools starting either from interactions data – the edges of the GRN –, or from the experimental data.

[3]Only the ones for which the `Effect` is different from 0.

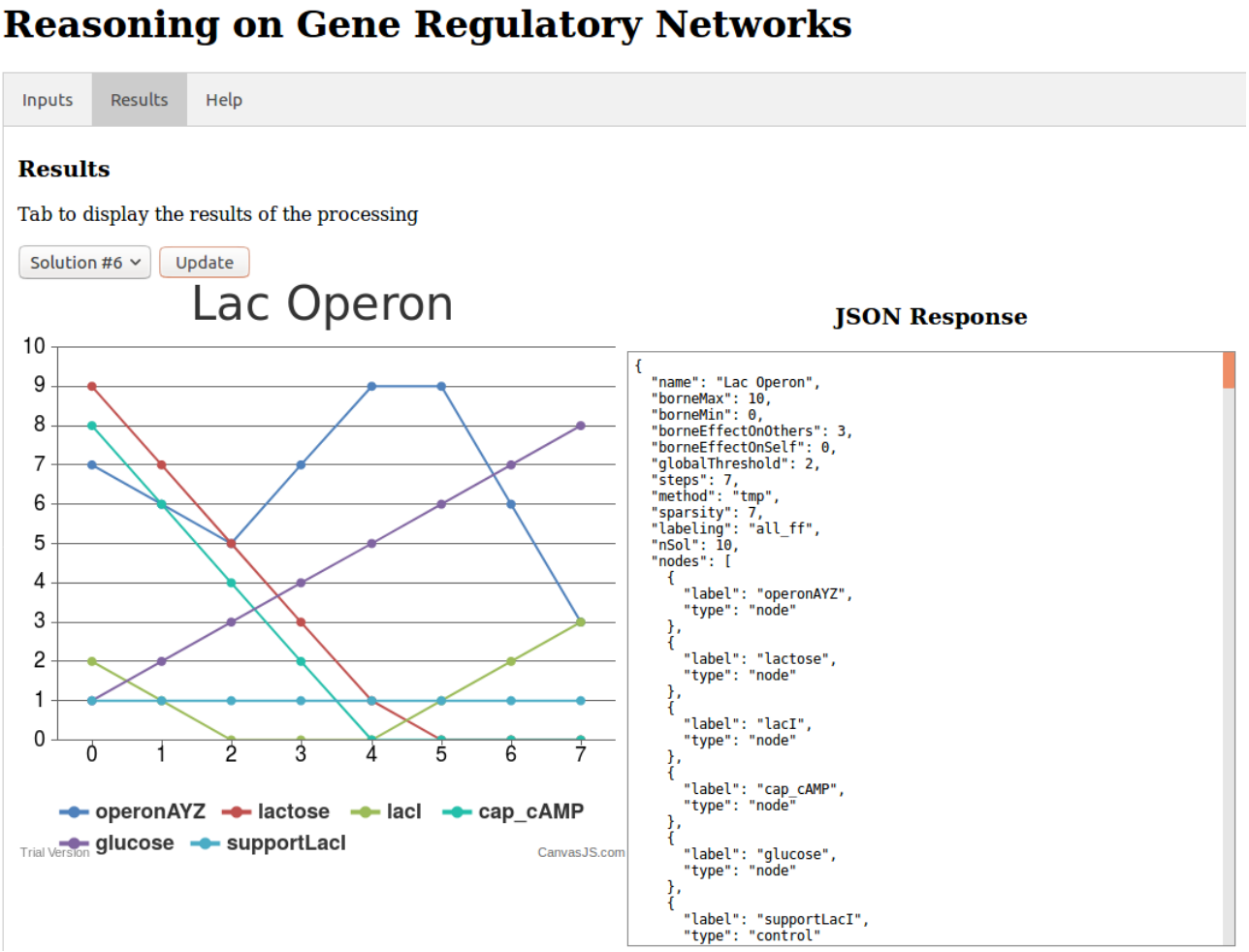

FIGURE 7.6: Print screen of the Results tab for the complete experiment

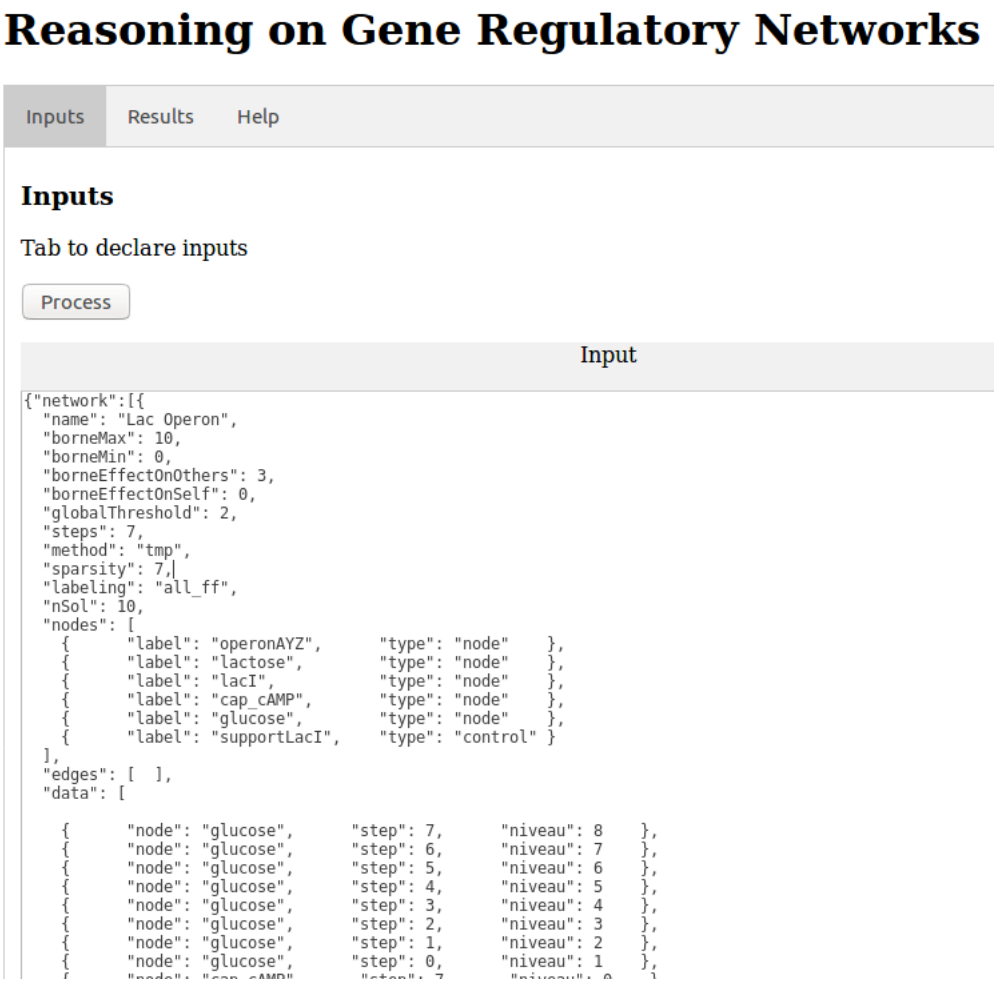

FIGURE 7.7: Print screen of the Inputs tab for the data inference: no edge is provided as input

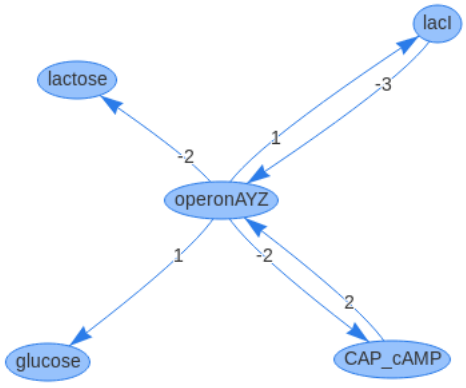

FIGURE 7.8: Network inferred based on provided experimental data

Chapter 8

Perspectives

8.1 Introduction

The work described in the previous chapters has opened many doors that could induce further studies. This chapter aims at compiling those perspectives, and to suggest avenues for reflection.

A first section describes the modeling opportunities opened in the context of constraint logic programming representation, mentioning other libraries and other techniques that can be used. A second section further discusses the constraint logic program itself: the performances, the choices of implementation and the portability of the current solution. The global architecture is succinctly reviewed, with suggestions on how to develop a real web application – production-grade – based on the current prototype tool. Finally, discussions about the test strategies will make the link with the biological world, starting point of this whole thesis, while suggesting other uses of the modeling and techniques presented.

8.2 Modeling

The model behind the current implementation is based on Thomas' work about boolean networks, extended to multi-valued. This is longly discussed in Chapter 1 (p3) and 3 (p33).

As also presented, different models exist and could lead to other constraints and representation. Per se, the use of CSP in the context of regulatory networks is not limited to one specific modeling. The current work could hence be extended to integrate other viewpoints.

As a first step, a good candidate is the linearized differential equations model. Written as constraints, those equations can be used to compute the concentration levels directly. A major impact however concerns the Finite Domain hypotheses: as such, the FD cannot be used directly as it is unlikely that solutions using the linearized differential equations will be found with integers only, at least without further precautions. Several tactics can be suggested:

- Keep the Finite Domain libraries in order to benefit from the built-in predicates, but consider coefficients on a much larger domain: instead of $[0, 10]$, one can use $[0, 1000]$, and rounding the values obtained to the nearest integer. This can be a *quick-win* solution in regards of the benefit of keeping a FD library in the CSP.

- Using real numbers libraries. Some constraints solvers have developed constrained over real numbers, that could be used as well. The labeling will then need to be modified: an enumeration of all possibilities can no longer be applied, as the domain is infinite. A home-made labeling strategy using a discretized domain can then be used to converge towards a solution. This is most likely to be combined with values rounding.

Another critical part is the asynchronous behavior modeling. In the current implementation, the representation could benefit from further work. The modeling of the asynchronous behavior, as introduced in Chapter 3 (p33), leads to two different tracks that should be discussed:

- Modification of the interactions pathway: instead of considering cyclic interactions, one can implement that for each step N, for each gene g of the system, are considered all the potential interactions i that could start, as the expression level of g is higher than the threshold θ_i. All those potential interactions are then sorted according to the Delay variable. The next interaction to occur is the one with the smallest `Delay` value.

- Considering the stochasticity of the events, one can also state that it is not possible to predict or estimate the order in which the interactions will take place. A stochastic behavior can therefore be introduced so that the next interaction is computed according to a statistic distribution.

Yet another consideration is related to the events that are modeled. As announced at the beginning in Section 1.2.3 (p5), although the main focus of this work is on the regulation during the transcription, other mechanisms happen, which all provide means to regulate the gene expression. Those post-transcription mechanisms are currently biologically less well explored, while it is believed that their effects may be as prevalent as the transcriptional control on the genes expressions. That is, while a gene may have no effect on the expression of another at RNA transcription level, it could be important for the protein expression, and be integrated in future versions of the tool.

8.3 CSP implementation

The implementation of the CSP can also be further developed, considering the modeling or the considerations elicited in Section 8.2 (p105).

Several improvement can be implemented:

- The integration of the self effect. The `association list` is already foreseen, and the impacts are only in the constraints linked to the `Niveau` variables, as described in Section 5.4.6 (p65).

- The `memoisation` does not yet integrate the `Delay` provided by the user, as confirmed in Section 5.5.5.2 (p83). Only the 0 value is added. The impact is limited to this one predicate only.

- The use of the `Delay` variable is currently limited. The user can provide a `Delay` value, but the system is not capable yet of inferring different `Delay` values. The impact of the modification would also be on the constraints specified in Section 5.4.6 (p65),

- The `ID` of an interaction is currently not used at all by the server. What's more, the server changes the value given to a default so that the user realizes it may have changed. However, this behavior is not optimal and the `ID` provided should be used and kept as reference of the interaction.

Those changes represent work on the current implementation but do not induce at first major impacts.

Other optimizations could be done on the CSP implementation in order to improve specifically the performances. Firstly, improved pruning and constraint propagation techniques could be used. Chapter 2 (p19) references types of constraints that can help pruning at an earlier stage of

the search process, such as the nogoods. Those techniques have not yet been used extensively. In the same idea, another search algorithm different from the backtracking, should be suggested. As confirmed in many sources such as [52], forward-checking or look-ahead techniques could be implemented so that the propagation and pruning of the search tree are more efficient, leading to improved performances.

Secondly, as already prepared, started and suggested, a relevant and global benchmark of the performances of execution regarding the type of user inputs should be performed, in order to apply the *best* resolution method or model case by case. The system could automatically select the most appropriate method to provide the desired results in the most efficient way.

Finally, the CSP as implemented, although using ISO Prolog as much as possible, is quite heavily linked to SWIPL and related implementation. To gain in portability and abstraction, it could be an interesting upgrade to move to a more general language to describe the constraints, such as suggested with the ZincMini approach (see [40]). It would also allow to benchmark the execution of different implementations on different platforms.

8.4 Production-Grade web application

As quite heavily discussed and specified, the current development results in a tool prototype which is not a production-grade software. To evolve to a usable and widely spread tool, several changes are required. The impacts are both on the front-end, obviously, and on the back-end.

8.4.1 Back-end

The current architecture and technology choices result in having the complete back-end handled in Prolog. Although it works well for limited scope and specific uses such as the prototype described, a production-grade Web Application requires much more from the back-end side.

As discussed in Chapter 4 (p41), the quality attributes have not really been taken into account. What's more, in this implementation, the Web Service and complete application back-end are essentially the same thing. Considering a layered architecture as an approved guideline, a production-grade Web Application would widely benefit from having an intermediate server in between the front-end and the Web Service in Prolog.

This intermediate server would concentrate the following responsibilities:

- Security aspects related to the data integrity and confidentiality. Other use cases could emerge from a more detailed security risks analysis,
- User management, to give the user the ability to retrieve a previous request for instance,
- Data persistence, with connections to a database (prerequisite for many use cases not implemented),
- Validation of the user inputs, as a second layer (after front-end), improving robustness and reliability,
- Connection to the CSP, using the JSON interface developed,
- Cache of the received response, part of the REST guidelines,
- Integration of other business cases over the inferred network post-processing

The technologies to use for this intermediate server are introduced in Chapter 4 (p41). As confirmed by the here above list, the possibilities of extended use cases are numerous.

8.4.2 Front-End

Currently, the front-end consists of a modern idea of single page application. The interface developed is restricted in terms of dynamism features offered to the user. It has not yet been fully developed for optimizing user experience.

This could be *easily* solved with a solid front-end development. The choice of architecture followed fully support the separation of concerns, leading to a completely unrelated front-end development with respect to the server-side.

A front-end using a modern framework as discussed in Chapter 4 (p41) could allow to improve the user experience, and develop many features:

- The way inputs are provided can be reviewed. The JSON file does not need to be known or viewed by the user: the inputs can be arranged as a JSON file in the background, while the user has a user-friendly interface to enter the data.

- More than only graphical improvements, the front-end can allow dynamic parameters modifications to respect the business related rules[1], automatic completion of parameters based on default set by the user, ... Sky is the limit regarding those kinds of facilitating features to implement in the front-end side.

- Verifications on the inputs can be performed before sending the request to the server. For instance, an incompatibility between parameters can be detected and either reported to the user or automatically corrected.

- The results can be expressed in the real form of a network, using dedicated network viewer libraries. Preliminary tests have been performed with **vis.js**[2] – all the network graphical representations in this thesis are realized using **vis.js** library –, quite promising although not yet finalized. The choice of library should however be more thoroughly investigated.

- Receiving the JSON file output from the back-end, the front-end can include several functionalities to help the user with the assessment of its hypothesis: comparison between graphs, responses filtering according to user-defined criteria,...

8.5 Uses

In a perspective of being used in a biological research context, a first step shall be to assess the developed tool on real cases. In close collaboration with biologists, real experiment data should be input to the tool, and the results and execution assessed in terms of relevance, performances, modeling, usability, etc.

The computer science tool shall remain a mean to help business-domain users to accomplish their tasks. In that way, new ideas will most likely come up from discussions with end users. This is one of the principles behind the Agile software development methods. Having set up a prototype with this tool, experiencing it with users would be a nice step further.

This tool has been developed in a specific context: the gene regulatory networks. Nothing however limits its use to this field of activities, provided the modeling fits another use case. Moreover, having set the modeling as an option for the user to select, adapting the tool to other network-related work is intrinsically within reach.

[1] For instance, the thresholds modified if the global limit is set lower

[2] **http://visjs.org/**.

8.6 Conclusion

This chapter summarizes the perspectives for future developments that this thesis raises, according to different topics discussed. Firstly, modeling improvements are suggested, either in terms of improving the current modeling based on logical method, or to integrate other modeling aspects such as linearized differential equations. Secondly, the CSP implementation is discussed: current limitations as the `Delay` inference, or gene self effect are listed. Ideas of optimizations regarding the performances, thanks to the developed systematic test framework, and the portability of the CSP are provided. Thirdly, the production-grade consideration is discussed: both back-end and front-end need further developments to become ready for production. It goes by quality attributes – such as usability and security – considerations, but also features such as data persistence implementation. Regarding the use of the tool developed, real cases applications can constitute a major step forward. Use cases outside of the gene regulatory networks inference can also be addressed, as long as the behind modeling is acceptable – knowing that this modeling is a user selected option, and several others can be implemented as well.

Conclusion

This thesis presents the use of constraint logic programming (CLP) paradigm in the context of the gene regulatory networks (GRN) inference, and introduces a tool prototype allowing to reason on GRN based on CLP.

The required theoretical background regarding the GRN, including the biological related notions is presented, including several modeling techniques commonly used. These techniques and their utility are discussed, and the concrete biological example of the lac operon is given. Then, the computer science related background related to the constraint logic programming paradigm is detailed. Some specificities of this technique of problem solving are explained, and the nominal resolution path is specified: variables elicitation, constraints elicitation, labeling.

Based on those two first chapters, the GRN inference problem is seen as a constraint satisfaction problem, and the three resolution steps adapted to GRN inference are detailed. This constitutes the base of the tool prototype.

The tool as a Web application is introduced starting from the general requirements, and its architecture and technological choices are discussed and motivated. The domain specific language created to describe the GRN itself and the inference tasks, including several parameters or options, is introduced. The implementation of the two sides of the Web application, back-end and front-end, is detailed. The back-end implementation, specifically, is finely analyzed and assessed, with the development of a systematic tests framework. With the help of this framework, different implementations suggested both for constraints modeling and labeling are evaluated. Promising results are given, suggesting further heuristic developments in order to improve the efficiency of the resolution.

Finally a case-based application, the lac operon, is studied with the help of the prototype developed. It gives the reader a concrete view of the use cases that are currently handled. While the outcomes are promising, further work is required.

Along the development of the tool, from the background up to the lac operon case-based application, several avenues for reflection and improvement are raised. They are compiled and discussed in the last chapter of this thesis, and summarized here after.

Firstly regarding the modeling itself, the main perspectives are the integration of other approved biological modelings, the modification of the asynchronous case modeling, and the integration of other biological effects on the regulation – the post-transcriptional mechanisms. A good candidate to integrate to the current modeling is the linearized differential equations model, which address quantitative aspects. It has however a major impact on the Finite Domain hypothesis. Different tactics are suggested: a *quick-win* technique allowing to keep the FD libraries, or a probably more complex one, with the integration of real numbers libraries in the constraints solving. The asynchronous modeling can be improved in two different suggested ways: reconsider the cyclic assumption, or integrate a stochastic behavior in the choice of application gene interaction.

Secondly, regarding the CSP implementation, several effects or interactions has not been implemented yet: the self effect – the interaction of a gene product concentration level with its own gene expression level –, the general inference of the `Delay`, or the global use of an identifier to uniquely refer to an interaction. The implementation can also be optimized regarding the CLP itself: other pruning, constraint propagation and search techniques, and – based on the systematic tests framework – the selection of the most appropriate implemented resolution method, considering the type of tasks to perform. The portability of this implementation is also discussed in Chapter 8 (p105).

Thirdly, in order to transform the prototype into a production-grade Web Application, and in particular integrating quality attributes, numerous opportunities and avenues are presented and discussed, for both back-end and front-end.

This thesis has been the opportunity to get familiarized with several concepts related to biology and computer science. Merging both domains have been incredibly interesting from a scientific perspective. As final words, a quote that guided all this work: *"Computer Science is a science of abstraction – creating the right model for a problem and devising the appropriate mechanizable techniques to solve it"*, [4].

Bibliography

[1] *A Beginner's Guide to Back-end Development*. URL: https://www.upwork.com/hiring/development/a-beginners-guide-to-back-end-development/ (visited on 08/16/2018).

[2] *A Guide to Stand-Alone Software*. URL: https://www.thebalance.com/types-of-stand-alone-software-1293731 (visited on 08/07/2018).

[3] Jamil Ahmad et al. *Hybrid Modelling and Dynamical Analysis of Gene Regulatory Networks with Delays*. 2006. DOI: 10.1159/000110010. URL: https://www.karger.com/Article/FullText/110010.

[4] Al. Aho and Jeff Ullman. *Foundations of Computer Science*. 1992. Chap. 1, The Mec, pp. 1–24. ISBN: 978-0716782841. URL: http://i.stanford.edu/{~}ullman/focs/ch01.pdf.

[5] *Beautiful HTML5 JavaScript Charts | CanvasJS*. URL: https://canvasjs.com/ (visited on 07/23/2018).

[6] Gilles Bernot et al. "Semantics of Biological Regulatory Networks". In: *Electronic Notes in Theoretical Computer Science* 180.3 (2007), pp. 3–14. ISSN: 15710661. DOI: 10.1016/j.entcs.2004.01.038.

[7] Hax Bradley. "Solving Linear Programs". In: *Applied mathematical programming*. Mathematic. 1. Addison-Wesley Publishing Company, 1977. Chap. 2, pp. 38–75.

[8] Fabien Corblin et al. *A declarative constraint-based method for analyzing discrete genetic regulatory networks*. 2009. DOI: 10.1016/j.biosystems.2009.07.007.

[9] *Cross-Origin Resource Sharing (CORS) - HTTP | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (visited on 08/08/2018).

[10] Frank Emmert-Streib, Matthias Dehmer, and Benjamin Haibe-Kains. "Gene regulatory networks and their applications: understanding biological and medical problems in terms of networks." In: *Frontiers in cell and developmental biology* 2 (2014), p. 38. ISSN: 2296-634X. DOI: 10.3389/fcell.2014.00038. URL: http://www.ncbi.nlm.nih.gov/pubmed/25364745http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4207011.

[11] Frank Emmert-Streib, Matthias Dehmer, and Benjamin Haibe-Kains. "Gene regulatory networks and their applications: understanding biological and medical problems in terms of networks." In: *Frontiers in cell and developmental biology* 2 (2014), p. 38. ISSN: 2296-634X. DOI: 10.3389/fcell.2014.00038. URL: http://www.ncbi.nlm.nih.gov/pubmed/25364745http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4207011.

[12] Vincent Englebert. *Software Architectures Engineering, Technologies and Methods-Engineering Method for Software Architectures*. URL: https://webcampus.unamur.be/course/view.php?id=809.

[13] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis. 2000, p. 162. ISBN: 0599871180. DOI: 10.1.1.91.2433. arXiv: arXiv:1011.1669v3. URL: http://www.ics.uci.edu/{~}fielding/pubs/dissertation/top.htm.

[14] Vladimir Filkov. *Identifying Gene Regulatory Networks from Gene Expression Data*. 2005. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3. URL: http://scholar.google.com/scholar?hl=en{\&}btnG=Search{\&}q=intitle:Identifying+Gene+Regulatory+Networks+from+Gene+Expression+Data+27.1{\#}0.

[15] Ferdinando Fioretto and Enrico Pontelli. *Constraint Programming in Community-Based Gene Regulatory Network Inference*. 2013.

[16] Raimo Franke, Fabian J Theis, and Steffen Klamt. "From binary to multivalued to continuous models: the lac operon as a case study." In: *Journal of integrative bioinformatics* 7.1 (2010), pp. 1–19. ISSN: 1613-4516. DOI: 10.2390/biecoll-jib-2010-151.

[17] Jonathan Fromentin et al. *Analysing gene regulatory networks by both constraint programming and model-checking*. 2007. DOI: 10.1109/IEMBS.2007.4353363.

[18] *Gene Expression: An Overview*. URL: https://www.news-medical.net/life-sciences/Gene-Expression-An-Overview.aspx (visited on 02/23/2018).

[19] *Gene Expression and Regulation — University of Leicester*. URL: https://www2.le.ac.uk/projects/vgec/highereducation/topics/geneexpression-regulation (visited on 02/23/2018).

[20] *Gene Expression Is Analyzed by Tracking RNA | Learn Science at Scitable*. URL: https://www.nature.com/scitable/topicpage/gene-expression-is-analyzed-by-tracking-rna-6525038 (visited on 07/21/2018).

[21] Anastasis Georgoulas, Jane Hillston, and Guido Sanguinetti. *ABC – Fun : A Probabilistic Programming Language for Biology*. 2013.

[22] Khaled Ghédira. *Constraint Satisfaction Problems*. 2009. DOI: 10.1016/j.jphysparis.2009.05.013. arXiv: 0803.2955.

[23] *global_cardinality/2*. URL: http://www.swi-prolog.org/pldoc/doc{_}for?object=global{_}cardinality/2 (visited on 08/11/2018).

[24] Fabrizio Grandoni and Giuseppe F Italiano. *Algorithms and Constraint Programming*. 2006.

[25] Anne Claire Haury et al. *TIGRESS: Trustful Inference of Gene REgulation using Stability Selection*. 2012. DOI: 10.1186/1752-0509-6-145. arXiv: 1205.1181.

[26] Jönsson Henrik. *Chapter 1 Modeling in systems biology*. URL: http://home.thep.lu.se/{~}henrik/fytn05/lectureNotesSysBiol.pdf.

[27] *Here are the best programming languages to learn in 2018*. URL: https://medium.freecodecamp.org/best-programming-languages-to-learn-in-2018-ultimate-guide-bfc93e615b35 (visited on 08/16/2018).

[28] Ryan Hoyle. *Overview: Gene regulation in bacteria*. 2017. URL: https://www.khanacademy.org/science/biology/gene-regulation/gene-regulation-in-bacteria/a/overview-gene-regulation-in-bacteria (visited on 03/01/2018).

[29] Vân Anh Huynh-Thu and Guido Sanguinetti. *Gene regulatory network inference: an introductory survey*. 2018. arXiv: 1801.04087. URL: http://arxiv.org/abs/1801.04087.

[30] Vân Anh Huynh-Thu et al. "Inferring regulatory networks from expression data using tree-based methods". In: *PLoS ONE* 5.9 (2010). Ed. by Mark Isalan, pp. 2009–2010. ISSN: 19326203. DOI: 10.1371/journal.pone.0012776. arXiv: arXiv:1205.1181v1. URL: http://dx.plos.org/10.1371/journal.pone.0012776.

[31] Masayo Inoue and Kunihiko Kaneko. "Cooperative Adaptive Responses in Gene Regulatory Networks with Many Degrees of Freedom". In: *PLoS Computational Biology* 9.4 (2013). Ed. by Erik van Nimwegen. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1003001. URL: http://dx.plos.org/10.1371/journal.pcbi.1003001.

[32] Jean-marie Jacquet. "Techniques d ' Intelligence Artificielle : Logic Programming". PhD thesis. 2016.

[33] Joxan Jaffart et al. "Constraint Logic Programming". In: *POPL '87 Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 1987, pp. 111–119. ISBN: 0897912152. DOI: 10.1145/234313.234416.

[34] *JSON (JavaScript Object Notation) Definition*. URL: https://techterms.com/definition/json (visited on 08/15/2018).

[35] George Katsirelos and F Bacchus. *Generalized nogoods in CSPs*. 2005. URL: http://www.aaai.org/Papers/AAAI/2005/AAAI05-062.pdf.

[36] S. A. Kauffman. "Metabolic stability and epigenesis in randomly constructed genetic nets". In: *Journal of Theoretical Biology* 22.3 (1969), pp. 437–467. ISSN: 10958541. DOI: 10.1016/0022-5193(69)90015-0. arXiv: NIHMS150003.

[37] Eric Lander, Robert Weinberg, and Claudette Gardel. *Introduction to Biology, Fall 2004* . 2004. URL: http://ocw.mit.edu (visited on 03/01/2018).

[38] A. Mandic et al. *A novel method for quantitative measurements of gene expression in single living cells*. 2017. DOI: 10.1016/j.ymeth.2017.04.008. URL: http://linkinghub.elsevier.com/retrieve/pii/S1046202316303632http://www.ncbi.nlm.nih.gov/pubmed/28456689.

[39] *Markov Assumption*. URL: https://www.cs.cmu.edu/{~}thrun/tutorial/sld030.htm (visited on 08/13/2018).

[40] *MiniZinc*. URL: http://www.minizinc.org/ (visited on 08/15/2018).

[41] *Modern Frontend Developer in 2018 – tajawal – Medium*. URL: https://medium.com/tech-tajawal/modern-frontend-developer-in-2018-4c2072fa2b9c (visited on 08/16/2018).

[42] *MVC architecture - App Center | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/Apps/Fundamentals/Modern{_}web{_}app{_}architecture/MVC{_}architecture (visited on 08/08/2018).

[43] *NFKB gene regulatory network*. URL: http://rulai.cshl.edu/TRED/GRN/NFKB.htm (visited on 08/12/2018).

[44] *No Title*. \url{https://projects.spring.io/spring-security/}.

[45] *Personas – A Simple Introduction | Interaction Design Foundation*. URL: https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them (visited on 08/06/2018).

[46] *Play Framework - Build Modern and Scalable Web Apps with Java and Scala*. URL: https://www.playframework.com/ (visited on 08/16/2018).

[47] *prolog - findall/3 creates new, unrelated variables in its resulting list - Stack Overflow*. URL: https://stackoverflow.com/questions/44728306/findall-3-creates-new-unrelated-variables-in-its-resulting-list (visited on 08/10/2018).

[48] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC - LS2N CNRS UMR 6241, COSLING S.A.S. 2017. URL: http://www.choco-solver.org.

[49] *Regulation of Gene Expression*. URL: https://www.news-medical.net/life-sciences/Regulation-of-Gene-Expression.aspx (visited on 02/23/2018).

[50] Adrien Richard, Jean-paul Comet, and Gilles Bernot. *R. Thomas' Logical Method*. 2008.

[51] Delphine Ropers et al. *Qualitative simulation of the carbon starvation response in Escherichia coli*. 2006. DOI: 10.1016/j.biosystems.2005.10.005.

[52] Francesca Rossi, Peter Van Beek, and Toby Walsh. "Constraint Programming". In: *Handbook of Constraint Programming*. Ed. by Francesca Rossi, Peter van Beek, and Toby Walsh. Elsevier, 2006. Chap. 1, pp. 1–31.

[53] Hana Rudová. *Constraint Programming and Scheduling Materials from the course taught at HTWG Constanz , Germany Constraint Programming and Scheduling : Outline Constraint Programming Constraint-based Scheduling*. 2009.

[54] *Scale-free networks - Math Insight*. URL: https://mathinsight.org/scale{_}free{_}network (visited on 07/23/2018).

[55] Thomas Schlitt and Alvis Brazma. *Current approaches to gene regulatory network modelling*. 2007. DOI: 10.1186/1471-2105-8-S6-S9. URL: http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-S6-S9.

[56] *Spring Documentation*.

[57] Sandra L Spurgeon, Robert C Jones, and Ramesh Ramakrishnan. "High Throughput Gene Expression Measurement with Real Time PCR in a Microfluidic Dynamic Array". In: *PLOS ONE* 3.2 (2008), pp. 1–7. DOI: 10.1371/journal.pone.0001662. URL: https://doi.org/10.1371/journal.pone.0001662.

[58] *SWI-Prolog – CLP(B)*. URL: http://www.swi-prolog.org/pldoc/man?section=clpb (visited on 08/14/2018).

[59] *SWI-Prolog*. URL: http://www.swi-prolog.org/ (visited on 08/01/2018).

[60] *The Java® Virtual Machine Specification*. URL: https://docs.oracle.com/javase/specs/jvms/se8/html/index.html (visited on 08/07/2018).

[61] *The NF-kB Signaling Pathway - Creative Diagnostics*. URL: https://www.creative-diagnostics.com/The-NF-kB-Signaling-Pathway.htm (visited on 08/12/2018).

[62] René Thomas. "Boolean Formalization of Genetic Control Circuits". In: *J. theor. Biol.* 42 (1973), pp. 563–585.

[63] René Thomas. *Regulatory Networks Seen as Asynchronous Automata : A Logical Description*. 1991.

[64] René Thomas and Richard D'Ari. *Biological Feedback*. Ed. by CRC Press. 1990, p. 316. ISBN: 0849367662.

[65] *Top JavaScript Frontend Frameworks Comparison in 2018 | FusionCharts*. URL: https://v3.fusioncharts.com/resources/js-frontend-frameworks-comparison/{\#}conclusion (visited on 08/16/2018).

[66] *Top JavaScript Libraries & Tech to Learn in 2018 – JavaScript Scene – Medium*. URL: https://medium.com/javascript-scene/top-javascript-libraries-tech-to-learn-in-2018-c38028e028e6 (visited on 08/16/2018).

[67] Markus Triska. *Constraint Logic Programming over Finite Domains*. URL: https://github.com/triska/clpfd (visited on 08/04/2018).

[68] Markus Triska. *The boolean constraint solver of SWI-prolog*. 2016. DOI: 10.1007/978-3-319-29604-3_4.

[69] a. Tsinober. *Constraint Logic Programming - An informal introduction*. 2001. DOI: 10.1007/0-306-48384-X. URL: http://www.springer.com/book/978-1-4020-0166-6.

[70] *Tutorial — Creating Web Applications in SWI-Prolog*. URL: http://www.pathwayslms.com/swipltuts/html/index.html (visited on 08/08/2018).

[71] *User Stories: An Agile Introduction*. URL: http://www.agilemodeling.com/artifacts/userStory.htm (visited on 08/06/2018).

[72] Van Anh Huynh-Thu. *GENIE3 vignette*. 2018. URL: https://bioconductor.org/packages/release/bioc/vignettes/GENIE3/inst/doc/GENIE3.html (visited on 08/15/2018).

[73] Nedumparambathmarath Vijesh, Swarup Kumar Chakrabarti, and Janardanan Sreekumar. "Modeling of gene regulatory networks: A review". In: *Journal of Biomedical Science and Engineering* 06.02 (2013), pp. 223–231. ISSN: 1937-6871. DOI: 10.4236/jbise.2013.62A027. URL: http://www.scirp.org/journal/PaperInformation.aspx?PaperID=28365{\&}{\#}abstract.

[74] W3. *AJAX Introduction*. URL: https://www.w3schools.com/xml/ajax{_}intro.asp (visited on 08/01/2018).

[75] W3. *CSS Tutorial*. URL: https://www.w3schools.com/css/default.asp (visited on 08/01/2018).

[76] W3. *HTML5 Introduction*. URL: https://www.w3schools.com/html/html5{_}intro.asp (visited on 08/01/2018).

[77] W3. *JavaScript Tutorial*. URL: https://www.w3schools.com/js/default.asp (visited on 08/01/2018).

[78] *Web Service Definition*. URL: https://techterms.com/definition/web{_}service (visited on 08/15/2018).

[79] *What is a gene? - Genetics Home Reference - NIH*. URL: https://ghr.nlm.nih.gov/primer/basics/gene (visited on 07/21/2018).

[80] *What is a Web Application?* URL: https://www.maxcdn.com/one/visual-glossary/web-application/ (visited on 08/07/2018).

[81] *What is DNA? - Genetics Home Reference*. URL: https://ghr.nlm.nih.gov/primer/basics/dna (visited on 02/23/2018).

[82] *What is Prolog? - Definition from Techopedia*. URL: https://www.techopedia.com/definition/24549/prolog (visited on 08/06/2018).

[83] *What is the correct quote for the Markov assumption? - Quora*. URL: https://www.quora.com/What-is-the-correct-quote-for-the-Markov-assumption (visited on 08/13/2018).

[84] *XML Introduction*. URL: https://www.w3schools.com/xml/xml{_}whatis.asp (visited on 08/15/2018).

[85] *XML vs JSON*. URL: https://www.cs.tufts.edu/comp/150IDS/final{_}papers/tstras01.1/FinalReport/FinalReport.html (visited on 08/15/2018).

[86] Qing Zhou et al. "A gene regulatory network in mouse embryonic stem cells." In: *Proceedings of the National Academy of Sciences of the United States of America*. Vol. 104. 42. National Academy of Sciences, 2007, pp. 16438–43. DOI: 10.1073/pnas.0701014104. URL: http://www.ncbi.nlm.nih.gov/pubmed/17940043http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2034259.

Appendices

Appendix A

Source Code

This appendix provides part of the source code. The complete resources and source code are available upon request to the author.

A.1 Prolog server code

```
1  :- use_module(library(http/thread_httpd)).
2  :- use_module(library(http/http_dispatch)).
3  :- use_module(library(http/http_json)).
4  :- use_module(library(http/http_cors)).
5
6
7  :- include(json2go).
8  :- include(phase0).
9  :- http_handler(/, say_hi, []).
10
11
12 :- http_handler('/test', handle_test, [methods(get,post,options)]).
13 :- http_handler('/api', handle_api, []).
14 :- set_setting(http:cors, [*]).
15
16
17 server(Port):-
18   http_server(http_dispatch, [port(Port), timeout(5000)]).
19 stop_server(Port):-
20   http_stop_server(Port,[]).
21
22 say_hi(_Request) :-
23         format('Content-type: text/plain~n~n'),
24     format('Hello World!~n').
25
26 handle_test(Request) :-
27   option(method(options), Request), !,
28   cors_enable(Request, [ methods([post])]),
29   format('~n').
30
31 handle_test(Request) :-
32     option(method(post), Request), !,
33     http_read_json(Request, JsonTest),
34
35     open('log_17072018.txt', write, MS),
36     write(MS, JsonTest),
37     close(MS),
38     produceJson(
39       lineaire,
40           test_put,
41       100,
42       0,
43       10,
44       2,
45       5,
46       [node-test2,node-test3],
47       [gen_gen(test2,test3,3)],
48       [niveau(node,test2,0,0), niveau(node,test3,0,0)],
```

```
49          DictOut),
50      cors_enable,
51      reply_json(DictOut,[status(201)]).
52
53  handle_api(Request) :-
54      option(method(options), Request), !,
55      cors_enable(Request, [ methods([post])]),
56      format('~n').
57
58  handle_api(Request) :-
59      option(method(post), Request), !,
60      http_read_json(Request, JsonIn),
61      setup_call_cleanup(createLog,
62      handle_api_(JsonIn,DictOut),
63      closeLog),
64      cors_enable,
65      reply_json(DictOut,[status(201)]).
66
67  % SI la resolution Failed
68  handle_api(Request) :-
69      option(method(post), Request), !,
70      http_read_json(Request, JsonIn),
71      cors_enable,
72      reply_json(JsonIn,[status(400)]).
73
74  %https://github.com/mollerse/sudoku.pl/blob/master/server.pl
75  handle_api(_) :- throw(http_reply(server_error('Method␣not␣supported.␣Only␣POST.'))).
76
77  handle_api_(JsonIn,Jsons) :-
78    myTranslate(Name,
79          BorneMaxNiveau,
80          BorneMinNiveau,
81          BorneEffectOnOthers,
82          BorneEffectOnSelf,
83          GlobalThreshold,
84          Steps,
85          Method,
86          Sparsity,
87          Labeling,
88          NSol,
89          Nodes,
90          Edges,
91          Data,
92          JsonIn),
93      multiple_instances(Name,Method,Sparsity,Labeling,NSol,Nodes, Edges,Data,BorneMinNiveau,
            BorneMaxNiveau,BorneEffectOnOthers,BorneEffectOnSelf,GlobalThreshold,Steps,Jsons).
94
95  % DictOut is the output as to be sent
96  one_instance(Name,Method,Sparsity,Labeling,NSol,Nodes, Edges,Data,BorneMinNiveau,
        BorneMaxNiveau,BorneEffectOnOthers,BorneEffectOnSelf,GlobalThreshold,Steps,DictOut):-
97    solve(Method,Sparsity,Labeling,Nodes, Edges,Data,BorneMinNiveau,BorneMaxNiveau,
          BorneEffectOnOthers,BorneEffectOnSelf,GlobalThreshold,Steps,Lassoc),
98    get_data_edge(Lassoc,Data_output,Edges_output),
99      produceJson(Name,
100     BorneMaxNiveau,
101     BorneMinNiveau,
102     BorneEffectOnOthers,
103     BorneEffectOnSelf,
104     GlobalThreshold,
105     Steps,
106     Method,
107     Sparsity,
108     Labeling,
109     NSol,
110     Nodes,
111     Edges_output,
112     Data_output,
113     DictOut).
114
115 multiple_instances(Name,Method,Sparsity,Labeling,NSol,Nodes, Edges,Data,BorneMinNiveau,
        BorneMaxNiveau,BorneEffectOnOthers,BorneEffectOnSelf,GlobalThreshold,Steps,Jsons):-
116       time(findnsols(NSol,DictOut,
117                 one_instance(Name,
118                       Method,
119                       Sparsity,
120                       Labeling,
```

```
121                         NSol,
122                         Nodes,
123                         Edges,
124                         Data,
125                         BorneMinNiveau,
126                         BorneMaxNiveau,
127                         BorneEffectOnOthers,
128                         BorneEffectOnSelf,
129                         GlobalThreshold,
130                         Steps,
131                         DictOut),
132                 DictOuts)),
133         build_final_json(DictOuts,Jsons).
134
135
136
137 solve(Method,Sparsity,Labeling,Nodes, Edges, Data, BorneMinNiveau,BorneMaxNiveau,
        BorneEffectOnOthers,BorneEffectOnSelf,GlobalThreshold,Steps,Lassoc):-
138             reseau(Method,Sparsity,Labeling,Nodes, Edges, Data, BorneMinNiveau,BorneMaxNiveau,
                    BorneEffectOnOthers,BorneEffectOnSelf,GlobalThreshold,Steps,Lassoc).
```

A.2 JSON from/to terms translation

```
1  :- use_module(library(http/json)).
2
3
4  %-------------------------------------------------------------------------%
5  %                                                                         %
6  %                          Lire un fichier JSON - test                    %
7  %                                                                         %
8  %-------------------------------------------------------------------------%
9  myRead :- open('network_test.json', read, MonStream),
10   (json_read(MonStream,Json),
11   write(Json),
12   %myTranslate(Json, ListeGene),
13   fail;true),
14   close(MonStream).
15
16 %-------------------------------------------------------------------------%
17 %                                                                         %
18 %                          JSON <-> PROLOG                                %
19 %                                                                         %
20 %-------------------------------------------------------------------------%
21
22 % mytranslate et produceJson pourraient etre un seul predicat!
23 myTranslate(Name,
24         BorneMaxNiveau,
25         BorneMinNiveau,
26         BorneEffectOnOthers,
27         BorneEffectOnSelf,
28         GlobalThreshold,
29         Steps,
30         Method,
31         Sparsity,
32         Labeling,
33         NSol,
34         Nodes,
35         Edges,
36         Data,
37         Json):-
38         Json = json([network=[
39         json([
40             name=Name,
41             borneMax=BorneMaxNiveau,
42             borneMin=BorneMinNiveau,
43             borneEffectOnOthers=BorneEffectOnOthers,
44             borneEffectOnSelf=BorneEffectOnSelf,
45             globalThreshold=GlobalThreshold,
46             steps=Steps,
47             method=Method,
48             sparsity=Sparsity,
49             labeling=Labeling,
50             nSol=NSol,
51           nodes=NodesJson,
52           edges=EdgesJson,
```

```
53          data=DataJson
54          ])
55        ]]),
56        data_translate(DataJson,Data),
57        edge_translate(EdgesJson,Edges),
58        node_translate(NodesJson,Nodes).
59
60
61  produceJson(Name,
62        BorneMaxNiveau,
63        BorneMinNiveau,
64        BorneEffectOnOthers,
65        BorneEffectOnSelf,
66        GlobalThreshold,
67        Steps,
68        Method,
69        Sparsity,
70        Labeling,
71        NSol,
72        Nodes,
73        Edges,
74        Data,
75        Json):-
76
77        data_translate(DataJson,Data),
78        edge_translate(EdgesJson,Edges),
79        node_produce(NodesJson,Nodes),
80        Json = json([
81            name=Name,
82            borneMax=BorneMaxNiveau,
83            borneMin=BorneMinNiveau,
84            borneEffectOnOthers=BorneEffectOnOthers,
85            borneEffectOnSelf=BorneEffectOnSelf,
86            globalThreshold=GlobalThreshold,
87            steps=Steps,
88            method=Method,
89            sparsity=Sparsity,
90            labeling=Labeling,
91            nSol=NSol,
92          nodes=NodesJson,
93          edges=EdgesJson,
94          data=DataJson
95          ]).
96
97
98
99  % [json([label=gene1,type=node]),json([label=gene2,type=node]),json([label=substA,type=control
        ])] -> [control-substA,node-gene2,node-gene1]
100 node_translate(Json,ListeGene):-
101   node_translate_acc(Json,ListeGene,[]),!.
102
103 node_translate_acc([],ListeGene,ListeGene).
104 node_translate_acc([H|T],ListeGene, Acc):-
105   H=json([label=Gene,type=node]),
106   NewAcc = [node-Gene|Acc],
107   node_translate_acc(T,ListeGene,NewAcc).
108 node_translate_acc([H|T],ListeGene, Acc):-
109   H=json([label=Gene,type=control]),
110   NewAcc = [control-Gene|Acc],
111   node_translate_acc(T,ListeGene,NewAcc).
112
113 % [control-substA,node-gene2,node-gene1] -> [json([label=gene1,type=node]),json([label=gene2,
        type=node]),json([label=substA,type=control])]
114
115 node_produce(Json,ListeGene):-
116   node_produce_(Json,ListeGene,[]),!.
117
118 node_produce_(Json, [], Json).
119 node_produce_(Json, [node-Gene|T], Acc):-
120   node_produce_(Json,T,[json([label=Gene, type=node])|Acc]).
121 node_produce_(Json, [control-Gene|T], Acc):-
122   node_produce_(Json,T,[json([label=Gene, type=control])|Acc]).
123
124
125 data_translate(Json,ListeData):-
126   data_translate_acc(Json,ListeData,[]).
```

```
127
128 data_translate_acc([],ListeData,ListeData):-!.
129 data_translate_acc([H|T], ListeData, Acc):-
130   H=json([node=Gene,step=Step,niveau=Niveau]),
131   NewAcc = [niveau(Gene,Step,Niveau)|Acc],
132   data_translate_acc(T,ListeData,NewAcc).
133
134
135
136 edge_translate(Json,ListeEdges):-
137   edge_translate_acc(Json,ListeEdges,[]),!.
138
139 edge_translate_acc([],ListeEdges,ListeEdges).
140 edge_translate_acc([H|T], ListeEdges, Acc):-
141   H=json([id=Id,from=GeneA,to=GeneB,threshold=Threshold, borneEffect=BorneEffect, delay=Delay,
        effect=Effect]),
142   NewAcc = [gen_gen(Id,GeneA,GeneB,Threshold,BorneEffect,Delay,Effect)|Acc],
143   edge_translate_acc(T,ListeEdges,NewAcc).
144
145
146 % To integrate several inner Json in the largest one
147 build_final_json(DictOuts,Jsons):-
148   Jsons = json([network=DictOuts]).
```

A.3 Main CLP

```
1  :- use_module(library(clpfd)).
2  :- include('./logger.pl').
3  :- include('./association.pl').
4  :- include('./convert_gene_list_to_rate_list.pl').
5  :- include('./contraintes_utilisateurs.pl').
6  :- include('./contraindre_lineaire.pl').
7  :- include('./contraindre_memoisation.pl').
8  :- include('./contraindre_tmp.pl').
9  :- include('./contraindre_asynchrone.pl').
10 :- include('./sparsity.pl').
11 :- include('./myLabeling.pl').
12
13
14 :- set_prolog_flag(encoding, utf8).
15
16 % ------------------------------------------------------------------------ %
17 %                                                                          %
18 %                          PREDICAT PRINCIPAL                              %
19 %                                                                          %
20 % ------------------------------------------------------------------------ %
21 reseau(Method,Sparsity,Labeling,Lgenes, Edges, Data,BorneMinNiveau, BorneMaxNiveau,
       BorneEffectOnOthers,BorneEffectOnSelf,GlobalThreshold,N,Lassoc) :-
22
23   logger("[reseau] - begin"),
24     produire_Liste_Assoc(Lgenes, steps(N),Levels, BorneMinNiveau,BorneMaxNiveau),
25     %logger("[reseau] - after niveau"),
26     produire_Liste_Assoc_gg(Lgenes,EffectOnOthers, BorneEffectOnOthers,GlobalThreshold,N),
27   %logger("[reseau] - after edge"),
28     produire_Liste_Assoc_self(Lgenes,EffectOnSelf, BorneEffectOnSelf),
29     %logger("[reseau] - after self"),
30     append([Levels,EffectOnOthers,EffectOnSelf],Lassoc),!,
31
32     logger1("[reseau] Lassoc = "), logger(Lassoc),
33
34     logger("[reseau] contraintes_utilisateurs - begin"),
35     % SpÃ©cifie les contraintes donnÃ©es par l'utilisateur (confiance : 100%)
36   contraintes_utilisateurs(Edges,Data,Lassoc),!,
37
38   logger("[reseau] contraintes_utilisateurs - end"),
39
40   % Fixer les niveaux non donnÃ©s pour le control
41   fixer_niveau_control(Lassoc),
42   !,% on ne veut pas offrir de backtracking avant ici,
43   logger1("[reseau] Avec Contraintes Utilisateurs et control fixÃ© - Lassoc = "), logger(
       Lassoc),
44   % Only Active when an Effect is set to 0,
45   prune_tree_effect0(Lassoc),
46   % logger1("[reseau] Prune Effect 0"), logger(Lassoc),
```

```
47    !,
48    printDomainLassoc(Lassoc),
49
50      logger("AVANT contraintes"),
51      %writeln(contraintes_niveaux),
52
53
54      get_time(Time_init),
55    sparsity(Sparsity,BorneEffectOnOthers,Lassoc),!,
56    get_time(Time_end),
57      Time_Sparsity is Time_end - Time_init,
58      logger1("[reseau] Temps Sparsity => "), logger(Time_Sparsity),
59
60    get_time(Time_init_con),
61    contraindre(Method,N,BorneMinNiveau,BorneMaxNiveau,Lassoc),
62    get_time(Time_end_con),
63      Time_Constrain is Time_end_con - Time_init_con,
64      logger1("[reseau] Temps Contraindre => "), logger(Time_Constrain),
65
66    get_time(Time_init_label),
67    my_labeling(Labeling,Lassoc),
68    get_time(Time_end_label),
69      Time_Label is Time_end_label - Time_init_label,
70      logger1("[reseau] Temps Labeling => "), logger(Time_Label),
71    impression(Lassoc).
72
73
74  % ---------------------------------------------------------------------- %
75  %                                                                        %
76  %                              CONTRAINTES                               %
77  % But : rÃ©duire les domaines de contraintes                             %
78  % ---------------------------------------------------------------------- %
79
80  contraindre(asynchronous,N,BorneMinNiveau,BorneMaxNiveau,Lassoc):-
81    contraindre_asynchrone_main(N,BorneMinNiveau,BorneMaxNiveau,Lassoc).
82  contraindre(memoisation,N,BorneMinNiveau,BorneMaxNiveau,Lassoc):-
83    contraindre_memoisation(Lassoc,BorneMinNiveau,BorneMaxNiveau).
84  contraindre(tmp,_N,BorneMinNiveau,BorneMaxNiveau,Lassoc):-
85    contraindre_tmp(Lassoc,BorneMinNiveau,BorneMaxNiveau,Lassoc).
86  contraindre(lineaire,_N,BorneMinNiveau,BorneMaxNiveau,Lassoc):-
87    contraindre_lineaire(Lassoc,BorneMinNiveau,BorneMaxNiveau,Lassoc).
88
89
90  boundaries_not_opti(In, In, Min, Max):-
91    In #=< Max,
92    In #>= Min.
93
94  boundaries_not_opti(In,Max,_,Max):-
95    In #> Max.
96  boundaries_not_opti(In,Min,Min,_):-
97    In #< Min.
98
99
100 boundaries(In,Out,Min,Max):-
101   V1 #= min(In,Max),
102   Out #= max(V1,Min).
103
104 % Si Effect est deja assignÃ© Ã  0, force le Threshold et la BorneEffect Ã  valeur min du
      domaine.
105 % Delay n'est pas forcÃ© pour compatibilitÃ© avec asynchronous
106
107 prune_tree_effect0([]).
108 prune_tree_effect0([gen_gen(_,_,_,Threshold,BorneEffect,_Delay,Effect)|Tail]):-
109   integer(Effect),
110   Effect #= 0,
111   fd_inf(Threshold,MinThreshold), Threshold#=MinThreshold,
112   fd_inf(BorneEffect,MinBorneEffect), BorneEffect#=MinBorneEffect,
113   %fd_inf(Delay,MinDelay), Delay#=MinDelay,
114   prune_tree_effect0(Tail).
115 prune_tree_effect0([gen_gen(_,_,_,_,_,_,_)|Tail]):-
116   prune_tree_effect0(Tail).
117 prune_tree_effect0([niveau(_,_,_,_)|Tail]):-
118   prune_tree_effect0(Tail).
119 prune_tree_effect0([self(_,_)|Tail]):-
120   prune_tree_effect0(Tail).
121
```

```
122
123
124 % ------------------------------------------------------------------------ %
125 %                                                                          %
126 %                                 UTILS                                    %
127 %                                                                          %
128 % ------------------------------------------------------------------------ %
129
130
131 valeur_absolue(ListeIn, ListeOut):-
132   valeur_absolue_(ListeIn,ListeOut,[]).
133 valeur_absolue_([],L,L).
134 valeur_absolue_([H|T],L,Acc):-
135   Tmp #= abs(H),
136   valeur_absolue_(T,L,[Tmp|Acc]).
137
138
139
140 %% Construit tous les couples possible d'une liste X donnÃ©e.
141 %% si X = [L1,L2,L3], R=[[L1,L2], [L1,L3], [L2,L3]]
142 couplePossible(X,R):- couplePossibleRec(X,[],R).
143 couplePossibleRec([_A|[]],R,R).
144 couplePossibleRec([H|T],R,Z):-
145   construireCouple([H|T],Rbis),
146   append(Rbis,R,Rnew),
147   couplePossibleRec(T,Rnew,Z).
148
149 %% Construit les couples formÃ©s par le premier Ã©lÃ©ment d'une liste, et tous les autres;
150 %% si L = [L1,L2,L3], Z = [[L1,L2],[L1,L3]]
151 construireCouple([H|T],Z):- construireCoupleRec([H|T],T,[],Z).
152 construireCoupleRec(_L,[],Z,Z).
153 construireCoupleRec([H|_],[Q|S],R,Z):-
154   construireCoupleRec([H|_],S,[[Q,H],[H,Q]|R],Z).
155
156 %% removeCouple : enleve les couples d'une liste de base impossible Ã  avoir.
157 removeCouple(Lbase,G,Lupdated) :- removeCoupleRec(Lbase,G,[],Lupdated).
158 removeCoupleRec([],_G,Z,Z).
159 removeCoupleRec([[X1,_]|T],G,Z,Lupdated):- not(member(X1,G)),removeCoupleRec(T,G,Z,Lupdated).
160 removeCoupleRec([[_,X2]|T],G,Z,Lupdated):- not(member(X2,G)),removeCoupleRec(T,G,Z,Lupdated).
161 removeCoupleRec([[X1,X2]|T],G,Z,Lupdated):- member(X1,G),member(X2,G),removeCoupleRec(T,G,[[X1
        ,X2]|Z],Lupdated).
162
163
164 getNiveau(Gene,Temps,Variable,Lassoc):-
165   member(niveau(_,Gene,Temps,Variable),Lassoc).
166
167 % a partir d'une liste de gene [geneA, geneB, geneC], retourne la liste des tuples:
168 % [[Niveau_geneA,Effect_geneA], [Niveau_geneB, Effect_geneB], [Niveau_geneC, Effect_geneC]].
169 getNiveauEffect(ListeInput, Temps, GeneCible, Lassoc, Output):-
170   getNiveauEffectAcc(ListeInput, Temps, GeneCible, Lassoc, Output, []).
171 getNiveauEffectAcc([],_,_,_,Out, Out).
172 getNiveauEffectAcc([H|T], Temps, GeneCible, Lassoc, Output, Acc):-
173   getNiveau(H,Temps,Niveau,Lassoc),
174   member(gen_gen(_,H,GeneCible,_,_,_,Effect), Lassoc),
175   Tmp = [Niveau,Effect],
176   NewAcc = [Tmp|Acc],
177   getNiveauEffectAcc(T,Temps,GeneCible,Lassoc,Output,NewAcc).
178
179
180 % ------------------------------------------------------------------------ %
181 %            Recuperation des DATA et EDGE Ã  partir de LASSOC              %
182 % ------------------------------------------------------------------------ %
183 get_data_edge(Lassoc,Data,Edge):-
184   get_data_edge_acc(Lassoc,Data,Edge,[],[]).
185
186 get_data_edge_acc([],Data,Edge,Data,Edge).
187
188 get_data_edge_acc([H|T], Data,Edge, DataAcc, EdgeAcc):-
189   H= niveau(_T,A,B,C),
190   DataAccNew = [niveau(A,B,C)|DataAcc],
191   get_data_edge_acc(T,Data,Edge,DataAccNew,EdgeAcc).
192
193 get_data_edge_acc([H|T], Data,Edge, DataAcc, EdgeAcc):-
194   H= gen_gen(_Id,_GeneA,_GeneB,_Threshold,_BorneEffect,_Delay,_Effect),
195   EdgeAccNew = [H|EdgeAcc],
196   get_data_edge_acc(T,Data,Edge,DataAcc,EdgeAccNew).
```

```
197
198 get_data_edge_acc([H|T], Data,Edge, DataAcc, EdgeAcc):-
199   H= self(_A,_B),
200    get_data_edge_acc(T,Data,Edge,DataAcc,EdgeAcc).
201
202
203
204 % ------------------------------------------------------------------------ %
205 %                                                                          %
206 %                          IMPRESSION    (modifie!)                        %
207 %                                                                          %
208 % ------------------------------------------------------------------------ %
209
210 impression([]) :- !.%, nl(stream).
211 impression([niveau(_,Gene,Step,V)|T]) :-
212   logger1('[impression]'),
213     format(atom(Text),'~a ~w ~d ~w ~a',[Gene,', t=',Step, ' -> ',V]),logger(Text),
214     logger_ice(Text),
215     impression(T).
216 impression([gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,V)|T]) :-
217   logger1('[impression]'),
218     format(atom(Text),'~a ~w ~a ~w ~d ~w ~d ~w ~w ~d',[GeneA,' -> ',GeneB, ' : ',V, '[',
          BorneEffect, ']',' th=', Threshold, ', delay= ', Delay]),logger(Text),
219     logger_ice(Text),
220     impression(T).
221 impression([self(Gene,V)|T]) :-
222   logger1('[impression]'),
223     format(atom(Text),'~a ~w ~d',[Gene,' -> ', V]),logger(Text),
224     logger_ice(Text),
225     impression(T).
226
227 %print(_).
228 print([]):-nl(stream).
229 print([H|T]):-
230   logger1("PRINT >"), nl(stream),
231   writeln(stream,H),
232   print(T).
233
234 %http://www.swi-prolog.org/pldoc/doc_for?object=fd_dom/2
235
236 printDomainLassoc([]).%:-nl(stream).
237 printDomainLassoc([niveau(_,Gene,Step,V)|T]) :-
238   logger1('[printDomainLassoc]'),
239   fd_dom(V,Dom),
240   format(atom(Text),'~w ~w ~d ~w ~w',[Gene,', t=',Step, ' -> ',V]),logger1(Text),
241   logger1('| domaine de Niveau -> '), logger1(V), logger1(' ==> '), logger1(Dom),nl(stream),
242     printDomainLassoc(T).
243 printDomainLassoc([gen_gen(_,GeneA,GeneB,Threshold,Borne,Delay,V)|T]) :-
244   logger1('[printDomainLassoc]'),
245     format(atom(Text),'~w ~w ~w ~w ~w',[GeneA,' -> ',GeneB, ' : ',V]),logger1(Text),
246     fd_dom(V,Dom),
247     fd_dom(Threshold,DomT),
248     fd_dom(Borne,DomB),
249     fd_dom(Delay,DomDelay),
250     logger1('| domaine Effect '), logger1(V), logger1(' ==> '), logger1(Dom),nl(stream),
251     logger1('| domaine Threshold  '), logger1(Threshold), logger1(' ==> '), logger1(DomT),nl(
          stream),
252     logger1('| domaine Borne  '), logger1(Borne), logger1(' ==> '), logger1(DomB),nl(stream),
253     logger1('| domaine Delay  '), logger1(Delay), logger1(' ==> '), logger1(DomDelay),nl(
          stream),
254     printDomainLassoc(T).
255 printDomainLassoc([self(Gene,V)|T]) :-
256   logger1('[printDomainLassoc]'),
257     format(atom(Text),'~w ~w ~w',[Gene,' -> ', V]),logger1(Text),
258     fd_dom(V,Dom),
259     logger1('| domaine de self Effect -> '), logger1(V), logger1(' ==> '), logger1(Dom),nl(
          stream),
260     printDomainLassoc(T).
```

A.3.1 Association list construction

```
1 % ------------------------------------------------------------------------ %
2 %                                                                          %
3 %                          LISTES ASSOCIATIVES                             %
4 %                                                                          %
5 % ------------------------------------------------------------------------ %
```

```
6
7
8  % ------------------------------------------------------------------------ %
9  %                              Gene - niveau                               %
10 % ------------------------------------------------------------------------ %
11
12
13 % on associe un gene avec
14 % - un temps
15 % - une variable (qui sera la concentration dans le domaine donnÃ©)
16 % - Nb est la borne du domaine pour toutes les concentrations
17 % steps(N) ou N est le nombre de steps considÃ©rÃ©s.
18 % produire_Assoc(ListeGenesInput, ListeAssoc, Nb), )
19 % BorneMinNiveau et BorneMaxNiveau sont des bornes globales
20 produire_Liste_Assoc(Input, steps(N),Output,BorneMinNiveau,BorneMaxNiveau):-
21   lassoc(Input, Output, N, N,BorneMinNiveau, BorneMaxNiveau).
22
23 lassoc([],[], _,_,_,_).
24 lassoc([Type-Gene|GenesTail], [niveau(Type,Gene,N,Niveau)|AssocTail], N, Nmax, BorneMinNiveau,
       BorneMaxNiveau):-
25   N>0,
26   NNew is N-1,
27   Niveau in BorneMinNiveau..BorneMaxNiveau,
28   lassoc([Type-Gene|GenesTail],AssocTail , NNew, Nmax, BorneMinNiveau,BorneMaxNiveau).
29 lassoc([Type-Gene|GenesTail], [niveau(Type,Gene,N,Niveau)|AssocTail], N, Nmax, BorneMinNiveau,
       BorneMaxNiveau):-
30   N=0,
31   Niveau in BorneMinNiveau..BorneMaxNiveau,
32   lassoc(GenesTail,AssocTail, Nmax, Nmax, BorneMinNiveau,BorneMaxNiveau).
33
34
35
36 % ------------------------------------------------------------------------ %
37 %                          Association gene Ã  gene                        %
38 % ------------------------------------------------------------------------ %
39
40
41 produire_Liste_Assoc_gg(Pairs,Output, BorneEffectOnOthers,GlobalThreshold,N):-
42   %pairs_values(Pairs, Input),
43   couplePossible(Pairs,ListeAssoc),
44   lassoc_gg(ListeAssoc,Output,BorneEffectOnOthers,GlobalThreshold,N).
45
46 % [[genA,genB],[genA,genC], [genB,genC]]
47 % devient
48 % [gen_gen(Id1,GeneA,GeneB,Threshold1,BorneEffect1,Effect1),
49 % gen_gen(Id2,GeneA,GeneC,Threshold2,BorneEffect2,Effect2),
50 % gen_gen(Id3,GeneB,GeneC,Threshold3,BorneEffect3,Effect3)]
51
52 lassoc_gg([],[],_,_,_).
53 % pas de lien sur une substance completement controlee
54 lassoc_gg([[T1-H1,control-H2]|T], [gen_gen(toChange,H1,H2,0,0, 0,0)| Acc], BorneEffectOnOthers
       ,GlobalThreshold,N ):-
55   lassoc_gg(T,Acc,BorneEffectOnOthers, GlobalThreshold,N).
56
57 lassoc_gg([[T1-H1,T2-H2]|T], [gen_gen(toChange,H1,H2,Threshold,BorneEffect, Delay,Effect)| Acc
       ], BorneEffectOnOthers,GlobalThreshold,N ):-
58
59   Threshold in 1..GlobalThreshold,
60   BorneEffect in 0 .. BorneEffectOnOthers,
61   abs(Effect) #=< BorneEffect,
62   Delay #>= 0,
63   Delay #< 1000*N, % purely arbitrary
64   lassoc_gg(T,Acc,BorneEffectOnOthers, GlobalThreshold,N).
65
66
67 % ------------------------------------------------------------------------ %
68 %                              Effet propre                                %
69 % ------------------------------------------------------------------------ %
70
71
72 produire_Liste_Assoc_self(Pairs, Output, BorneEffectOnSelf):-
73   pairs_values(Pairs, Input),
74   lassoc_self(Input,Output, BorneEffectOnSelf).
75
76 lassoc_self([],[],_).
77 lassoc_self([H|T], [self(H,Var)|Tail], BorneEffectOnSelf):-
```

```
78
79    abs(Var) #=< BorneEffectOnSelf,
80    lassoc_self(T, Tail, BorneEffectOnSelf).
```

A.4 Constraints predicates

A.4.1 User Constraints

```
1  %-------------------------------------------------------------------------%
2  %                                                                         %
3  %                          CONTRAINTES_UTILISATEURS/3                     %
4  % But : rÃ©duire les domaines de contraintes en fonction de l'entrÃ©e     %
5  % utilisateur                                                            %
6  %-------------------------------------------------------------------------%
7
8
9  %-------------------------------------------------------------------------%
10 % Edges => [ gen_gen(geneC,geneB,0) ,gen_gen(geneB,geneA,0)              ]
11 % gen_gen(Id,GeneA,GeneB,Threshold,BorneEffect,Delay,Effect)
12 % Data  => [ niveau(_,geneA,2,5),niveau(_,geneB,1,2),niveau(_,geneC,0,4) ]
13 %-------------------------------------------------------------------------%
14
15 % cas 0 : s'il n'y a plus de contraintes Ã  prendre en compte, on arrete
16 contraintes_utilisateurs([],[],_):-
17    !,
18    logger("Fin␣des␣contraintes␣Utilisateurs␣[]␣[]").
19
20 % cas 1 : un edge est spÃ©cifiÃ© et correspond Ã  l'entrÃ©e de Lassoc
21 % gen_gen(Id,H1,H2,Threshold,BorneEffect, Var)
22 % -> plusieurs cas Ã  considÃ©rer ==> il y a trois variables, et l'utilisateur peut donner
       indÃ©pendamment autant de combinaisons
23 % qu'il le souhaite:
24 % 1 - uniquement effet
25 % 2 - uniquement threshold
26 % 3 - uniquement borneEffect
27 % 4 - effet et threshold
28 % 5 - effet et borneEffect
29 % 6 - threshold et borneEffect
30 % 7 - effet, threshold et borneEffect
31
32 % 8 - uniquement delai
33 % 9 - threshold et delai
34 % 10- borneEffect et delai
35 % 11- effet et delai
36 % 12- effet et threshold et delai
37 % 13- effet et borneEffect et delai
38 % 14- threshold et borneEffect et delai
39 % 15- threshold et borneEffect et delai et effet
40
41 % 1- on fournit uniquement l'effect de la transition (Threshold et BorneEffect ne sont pas
       donnÃ©s)
42 % => le Threshold applicable est le GlobalThreshold
43 % => le BorneEffect applicable est le gÃ©nÃ©ral
44 contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
       TEdges], Data, Lassoc):-
45    member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
46    ThresholdGiven = '/',
47    BorneEffectGiven = '/',
48    DelayGiven = '/',
49
50    integer(X), fd_size(X,1),
51    Var #= X,
52    logger("[contraintes_utilisateurs]␣cas␣1"),
53    contraintes_utilisateurs(TEdges,Data,Lassoc).
54
55 % 2- on fournit uniquement le threshold de la transition
56 % => on laisse les domaines des autres variables tels quels
57 contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
       TEdges], Data, Lassoc):-
58    member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
59    % Le domaine de Var est dÃ©jÃ  limitÃ©
60    X = '/',
61    BorneEffectGiven = '/',
```

```
62    DelayGiven ='/',
63
64    integer(ThresholdGiven), fd_size(ThresholdGiven,1),
65    Threshold #= ThresholdGiven,
66    % BorneEffect est dÃ©jÃ  limitÃ© Ã  -BorneEffectOnOthers -> BorneEffectOnOthers
67    logger("[contraintes_utilisateurs] cas 2"),
68    contraintes_utilisateurs(TEdges,Data,Lassoc).
69
70  % 3- on fournit la borne de l'effect de la transition
71  contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
        TEdges], Data,Lassoc):-
72    member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
73    % Le domaine de Var est dÃ©jÃ  limitÃ©
74    % Le domaine de Threshold est dÃ©jÃ  limitÃ©
75    ThresholdGiven = '/',
76    X = '/',
77    DelayGiven ='/',
78
79    integer(BorneEffectGiven), fd_size(BorneEffectGiven,1),
80    BorneEffect#=BorneEffectGiven,
81    % Etant donnÃ© que l'on recontraint BorneEffect, on peut diminuer le domaine de l'Effect (
          Var)
82    %Lower is -BorneEffect,
83    %Higher is BorneEffect,
84    abs(Var) #=< BorneEffect,
85
86    logger("[contraintes_utilisateurs] cas 3"),
87    contraintes_utilisateurs(TEdges,Data,Lassoc).
88  % 4- effect et Threshold
89  contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
        TEdges], Data,Lassoc):-
90
91    member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
92    BorneEffectGiven = '/',
93    DelayGiven ='/',
94
95    integer(X), fd_size(X,1),
96    integer(ThresholdGiven), fd_size(ThresholdGiven,1),
97
98    Var #= X,
99    Threshold #= ThresholdGiven,
100
101   logger("[contraintes_utilisateurs] cas 4"),
102   contraintes_utilisateurs(TEdges,Data,Lassoc).
103
104 % 5- effect et borneEffect
105 contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
        TEdges], Data,Lassoc):-
106   member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay, Var), Lassoc),
107   ThresholdGiven = '/',
108   DelayGiven='/',
109   integer(X), fd_size(X,1),
110   integer(BorneEffectGiven), fd_size(BorneEffectGiven,1),
111
112   Var #= X,
113   BorneEffect#=BorneEffectGiven,
114   % Etant donnÃ© que l'on recontraint BorneEffect, on peut diminuer le domaine de l'Effect (
          Var)
115   % (double vÃ©rification car Var est dÃ©jÃ  unifiÃ©)
116   %Lower is -BorneEffect,
117   %Higher is BorneEffect,
118   abs(Var) #=< BorneEffect,
119   % Threshold est dÃ©jÃ  limitÃ© Ã  0 .. GlobalThreshold
120   logger("[contraintes_utilisateurs] cas 5"),
121   contraintes_utilisateurs(TEdges,Data,Lassoc).
122
123 % 6- Threshold et borneEffect
124 contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
        TEdges], Data,Lassoc):-
125   member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
126   X = '/',
127   DelayGiven = '/',
128   integer(ThresholdGiven), fd_size(ThresholdGiven,1),
129   integer(BorneEffectGiven), fd_size(BorneEffectGiven,1),
130
```

```
131     Threshold #= ThresholdGiven,
132     BorneEffect #= BorneEffectGiven,
133
134     % Etant donnÃ© que l'on recontraint BorneEffect, on peut diminuer le domaine de l'Effect (
            Var)
135     %Lower is -BorneEffect,
136     %Higher is BorneEffect,
137     %Var in Lower .. Higher,
138
139     abs(Var) #=< BorneEffect,
140     % X est dÃ©jÃ  limitÃ©
141     logger("[contraintes_utilisateurs] cas 6"),
142     contraintes_utilisateurs(TEdges,Data,Lassoc).
143
144   % 7- Effect, Threshold et borneEffect
145   contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
          TEdges], Data, Lassoc):-
146     member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
147     DelayGiven='/',
148     integer(X), fd_size(X,1),
149     integer(ThresholdGiven), fd_size(ThresholdGiven,1),
150     integer(BorneEffectGiven), fd_size(BorneEffectGiven,1),
151
152     Var #= X,
153     Threshold #= ThresholdGiven,
154     BorneEffect #= BorneEffectGiven,
155
156     % Etant donnÃ© que l'on recontraint BorneEffect, on peut diminuer le domaine de l'Effect (
            Var)
157     % (double vÃ©rification car Var est dÃ©jÃ  unifiÃ©)
158     %Lower is -BorneEffect,
159     %Higher is BorneEffect,
160
161     abs(Var) #=< BorneEffect,
162     % X est dÃ©jÃ  limitÃ©
163     logger("[contraintes_utilisateurs] cas 7"),
164     contraintes_utilisateurs(TEdges,Data,Lassoc).
165
166
167   % 8 - uniquement delai
168
169   contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
          TEdges], Data, Lassoc):-
170     member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
171     ThresholdGiven = '/',
172     BorneEffectGiven = '/',
173     X = '/',
174
175     integer(DelayGiven), fd_size(DelayGiven,1),
176     Delay #= DelayGiven,
177
178     logger("[contraintes_utilisateurs] cas 8"),
179     contraintes_utilisateurs(TEdges,Data,Lassoc).
180
181
182   % 9 - threshold et delai
183   contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
          TEdges], Data, Lassoc):-
184
185     member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
186     BorneEffectGiven = '/',
187     X ='/',
188
189     integer(DelayGiven), fd_size(DelayGiven,1),
190     integer(ThresholdGiven), fd_size(ThresholdGiven,1),
191
192     Delay #= DelayGiven,
193     Threshold #= ThresholdGiven,
194
195     logger("[contraintes_utilisateurs] cas 9"),
196     contraintes_utilisateurs(TEdges,Data,Lassoc).
197
198
199   % 10- borneEffect et delai
200   contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
          TEdges], Data, Lassoc):-
```

```
201
202     member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
203     ThresholdGiven = '/',
204     X ='/',
205
206     integer(DelayGiven), fd_size(DelayGiven,1),
207     integer(BorneEffectGiven), fd_size(BorneEffectGiven,1),
208
209     Delay #= DelayGiven,
210     BorneEffect #= BorneEffectGiven,
211
212     logger("[contraintes_utilisateurs] cas 10"),
213     contraintes_utilisateurs(TEdges,Data,Lassoc).
214
215  % 11- effet et delai
216  contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
         TEdges], Data,Lassoc):-
217     member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
218     ThresholdGiven = '/',
219     BorneEffectGiven = '/',
220
221     integer(DelayGiven), fd_size(DelayGiven,1),
222     integer(X), fd_size(X,1),
223
224     Delay #= DelayGiven,
225     Var #= X,
226
227     logger("[contraintes_utilisateurs] cas 11"),
228     contraintes_utilisateurs(TEdges,Data,Lassoc).
229
230  % 12- effet et threshold et delai
231  contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
         TEdges], Data,Lassoc):-
232     member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
233
234     BorneEffectGiven = '/',
235
236     integer(ThresholdGiven), fd_size(ThresholdGiven,1),
237     integer(DelayGiven), fd_size(DelayGiven,1),
238     integer(X), fd_size(X,1),
239
240     Delay #= DelayGiven,
241     Threshold #= ThresholdGiven,
242     Var #= X,
243
244     logger("[contraintes_utilisateurs] cas 12"),
245     contraintes_utilisateurs(TEdges,Data,Lassoc).
246
247  % 13- effet et borneEffect et delai
248  contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
         TEdges], Data,Lassoc):-
249     member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
250     ThresholdGiven = '/',
251
252     integer(BorneEffectGiven), fd_size(BorneEffectGiven,1),
253     integer(DelayGiven), fd_size(DelayGiven,1),
254     integer(X), fd_size(X,1),
255
256     Delay #= DelayGiven,
257     BorneEffect #= BorneEffectGiven,
258     Var #= X,
259
260     logger("[contraintes_utilisateurs] cas 13"),
261     contraintes_utilisateurs(TEdges,Data,Lassoc).
262
263  % 14- threshold et borneEffect et delai
264  contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
         TEdges], Data,Lassoc):-
265
266     member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
267
268     X ='/',
269
270     integer(ThresholdGiven), fd_size(ThresholdGiven,1),
271     integer(DelayGiven), fd_size(DelayGiven,1),
272     integer(BorneEffectGiven), fd_size(BorneEffectGiven,1),
```

```
273
274     Delay #= DelayGiven,
275     BorneEffect #= BorneEffectGiven,
276     Threshold #= ThresholdGiven,
277
278     logger("[contraintes_utilisateurs] cas 14"),
279     contraintes_utilisateurs(TEdges,Data,Lassoc).
280
281
282
283   % 15- threshold et borneEffect et delai et effet
284
285   contraintes_utilisateurs([gen_gen(_,GeneA,GeneB,ThresholdGiven,BorneEffectGiven,DelayGiven,X)|
          TEdges], Data, Lassoc):-
286
287     member(gen_gen(_,GeneA,GeneB,Threshold,BorneEffect,Delay,Var), Lassoc),
288
289
290     integer(X), fd_size(X,1),
291     integer(ThresholdGiven), fd_size(ThresholdGiven,1),
292     integer(DelayGiven), fd_size(DelayGiven,1),
293     integer(BorneEffectGiven), fd_size(BorneEffectGiven,1),
294
295     Delay #= DelayGiven,
296     BorneEffect #= BorneEffectGiven,
297     Threshold #= ThresholdGiven,
298     Var #= X,
299
300     logger("[contraintes_utilisateurs] cas 15"),
301     contraintes_utilisateurs(TEdges,Data,Lassoc).
302
303
304
305   % cas 2 : une data est spÃ©cifiÃ©e et correspond Ã  l'entrÃ©e de Lassoc
306   contraintes_utilisateurs(Edges, [niveau(Gene,Temps,X)|TData], Lassoc):-
307     getNiveau(Gene,Temps,Var,Lassoc),
308     Var #= X,
309     contraintes_utilisateurs(Edges,TData,Lassoc).
310
311   % cas 3 : ???
312   contraintes_utilisateurs(Edges,[H|T],Lassoc):-
313     logger("[contraintes_utilisateurs] ERROR "),
314     logger1("Edges : "), logger(Edges),
315     logger1("H:"), logger(H),
316     logger1("T:"), logger(T),
317     logger1("Lassoc:"), logger(Lassoc),
318     logger("Fin du cas 3"),
319     fail.
320   %
321
322
323
324   %-------------------------------------------------------------%
325   % But : fixer le threshold d'activation de la relation       %
326   % a une valeur donnÃ©e                                       %
327   %-------------------------------------------------------------%
328
329   fixer_threshold(_,[]).
330   fixer_threshold(GlobalThreshold,[niveau(_,_,_,_)|T]):-
331     fixer_threshold(GlobalThreshold,T).
332   fixer_threshold(GlobalThreshold,[self(_,_)|T]):-
333     fixer_threshold(GlobalThreshold,T).
334   fixer_threshold(GlobalThreshold,[gen_gen(_,_,_,Threshold,_,_,_)|T]):-
335     fd_size(Threshold,1),
336     fixer_threshold(GlobalThreshold,T).
337   fixer_threshold(GlobalThreshold,[gen_gen(_,_,_,Threshold,_,_,_)|T]):-
338     not(fd_size(Threshold,1)),
339     Threshold in GlobalThreshold,
340     fixer_threshold(GlobalThreshold,T).
341
342
343   %-------------------------------------------------------------%
344   % But : fixer la borne d'effet de la relation                %
345   % a une valeur donnÃ©e                                       %
346   %-------------------------------------------------------------%
347
```

```
348 fixer_borne(_,[]).
349 fixer_borne(BorneEffect,[niveau(_,_,_,_)|T]):-
350   fixer_borne(BorneEffect,T).
351 fixer_borne(BorneEffect,[self(_,_)|T]):-
352   fixer_borne(BorneEffect,T).
353 fixer_borne(BorneEffect,[gen_gen(_,_,_,_,Borne,_,_)|T]):-
354   fd_size(Borne,1),
355   fixer_borne(BorneEffect,T).
356 fixer_borne(BorneEffect,[gen_gen(_,_,_,_,Borne,_,_)|T]):-
357   not(fd_size(Borne,1)),
358   Borne #= BorneEffect,
359   fixer_borne(BorneEffect,T).
360
361 %-------------------------------------------------------------%
362 % But : fixer le delai de toutes les relations               %
363 % a une valeur donnÃ©e                                       %
364 % ?- fixer_delay(+Valeur,+Lassoc).                          &
365 %-------------------------------------------------------------%
366
367 fixer_delay(_,[]).
368 fixer_delay(DelayValue,[niveau(_,_,_,_)|T]):-
369   fixer_delay(DelayValue,T).
370 fixer_delay(DelayValue,[self(_,_)|T]):-
371   fixer_delay(DelayValue,T).
372 fixer_delay(DelayValue,[gen_gen(_,_,_,_,_,DelayValue,_)|T]):-
373   fixer_delay(DelayValue,T).
374 fixer_delay(DelayValue,[gen_gen(_,_,_,_,_,Delay,_)|T]):-
375   not(fd_size(Delay,1)),
376   Delay #= DelayValue,
377   fixer_borne(DelayValue,T).
378
379
380
381 %-------------------------------------------------------------%
382 % But : fixer l'effet de la relation                         %
383 % a une valeur donnÃ©e  - should not be used          %
384 %-------------------------------------------------------------%
385
386 fixer_effect([]):-
387   logger(fixer_effect_end).
388 fixer_effect([niveau(_,_,_,_)|T]):-
389   fixer_effect(T).
390 fixer_effect([self(_,_)|T]):-
391   fixer_effect(T).
392 fixer_effect([gen_gen(_,_,_,_,_,_,Effect)|T]):-
393   fd_size(Effect,1),
394   fixer_effect(T).
395 fixer_effect([gen_gen(_,_,_,_,_,_,Effect)|T]):-
396   not(fd_size(Effect,1)),
397   Effect #= 0,
398   fixer_effect(T).
399
400 %-------------------------------------------------------------%
401 % But : fixer les niveaux des substances de controle         %
402 %-------------------------------------------------------------%
403 fixer_niveau_control([]):-!.
404 fixer_niveau_control([niveau(control,_,_,Niveau)|T]):-
405   integer(Niveau),
406   fixer_niveau_control(T).
407 fixer_niveau_control([niveau(control,_,_,Niveau)|T]):-
408   not(integer(Niveau)),
409   Niveau #=0,
410   fixer_niveau_control(T).
411 fixer_niveau_control([niveau(node,_,_,_)|T]):-
412   fixer_niveau_control(T).
413 fixer_niveau_control([self(_,_)|T]):-
414   fixer_niveau_control(T).
415 fixer_niveau_control([gen_gen(_,_,_,_,_,_,_)|T]):-
416   fixer_niveau_control(T).
```

A.4.2 `Niveau` constraints methods

```
1 contraindre_lineaire([],_,_,_).
2 contraindre_lineaire([gen_gen(_,_,_,_,_,_,_)|Tail],BorneMinNiveau,BorneMaxNiveau,Lassoc):-
3   contraindre_lineaire(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc).
```

```
4  contraindre_lineaire([self(_,_)|Tail],BorneMinNiveau,BorneMaxNiveau, Lassoc):-
5    contraindre_lineaire(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc).
6
7  contraindre_lineaire([niveau(control,_,_,_)|Tail], BorneMinNiveau,BorneMaxNiveau, Lassoc):-
8    logger("[contraindre_lineaire] niveau control => not constrained"),
9    contraindre_lineaire(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc).
10
11 contraindre_lineaire([niveau(node,Gene,Temps,Var)|Tail],BorneMinNiveau,BorneMaxNiveau, Lassoc)
       :-
12   logger("--------------------contraindre_lineaire-----------------------"),
13   Temps>0,
14
15   TempsPrev #= Temps - 1,
16
17   logger1("[contraindre_lineaire] - Temps :"), logger(Temps),
18   logger1("[contraindre_lineaire] - Temps_Prev:"), logger(TempsPrev),
19   getNiveau(Gene,TempsPrev,VarPrev,Lassoc),
20   logger1("[contraindre_lineaire] - Var Prev :"), logger(VarPrev),
21   logger1("[contraindre_lineaire] - Var :"), logger(Var),
22   % lien entre une variable et le calcul
23   % VarCourante = VariablePrÃ©cÃ©dente + expressor_impact (simple addition)
24
25   % Liste des taux applicables sur la variable courante, calculÃ© uniquement pour les
         expresseurs actifs
26
27   % 1) recuperer les Expressors
28   findall(GeneFA, member(gen_gen(_,GeneFA,Gene,_,_,_,_), Lassoc), Expressors),
29
30   logger1("[contraindre_lineaire] - sur le gene :"), logger(Gene),
31   logger1("[contraindre_lineaire] - Expressors :"), logger(Expressors),
32
33   convertGeneListToRateList(Gene,Expressors,Lassoc,Temps,TauxList),
34   logger1("[contraindre_lineaire] - Liste Taux :"), logger(TauxList),
35
36   sum(TauxList,#=,Contribution),
37   logger1("[contraindre_lineaire] - Contribution :"), logger(Contribution),
38
39 %% %ODE% 3) recuperer les effets de ces expressors : Output = [[Niveau1,Effect1], [Niveau2,
       Effect2], ...]
40   %ODE% getNiveauEffect(Expressors, TempsPrev, Gene, Lassoc, NiveauEffect),
41 %% %ODE% 4) ResultTmp = Produit scalaires des deux vecteurs, avec les effects "1000" x trop
       grand
42   %ODE% sumAllList(NiveauEffect,ResultTmp),
43   %ODE% Contribution #= ResultTmp div 100, % 1000 car W(effects) sont pris entre 0 et 1000 ->
         0 et 1 => 500 ==> 0.5
44   %ODE% 5) Result = Result
45
46
47   Tmp #= VarPrev+Contribution,
48   logger("[contraindre_lineaire] - Apres Calcul Tmp #= VarPrev + Contribution"),
49   boundaries(Tmp,Var,BorneMinNiveau,BorneMaxNiveau),
50   logger1("[contraindre_lineaire] - Var = "), logger(Var),
51   logger("--------------------END Contribution -------------------"),
52
53   %ODE% Var #< Tmp + 10,
54   %ODE% Var #> Tmp - 10,
55
56   contraindre_lineaire(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc).
57
58 contraindre_lineaire([niveau(node,Gene,0,_)|Tail], BorneMinNiveau,BorneMaxNiveau, Lassoc):-
59   getNiveau(Gene,1,_,Lassoc),
60   logger("[contraindre_lineaire] niveau 0"),
61   contraindre_lineaire(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc).


1
2
3  %/**************************************************************************/
4  % convertGeneListToRateList
5  %
6  %/**************************************************************************/
7
8
9  %%% convertGeneListToVariableList(Input,Output).
10 %%% But : a partir d'une liste d'input (des genes expresseurs),
11 %%% on renvoie une liste des variables des niveaux correspondants.
12 %%% Il est nÃ©cessaire de donner le temps, et la liste d'association
```

```
13 %%convertGeneListToVariableList(Input,Lassoc,Temps,Output):-
14 %%   convertGeneListToVariableListAcc(Input,Lassoc,Temps,Output,[]).
15
16 %%convertGeneListToVariableListAcc([],_,_,Output,Output).
17 %%convertGeneListToVariableListAcc([H|T],Lassoc,Temps,Output,Acc):-
18 %%   getNiveauFromGene(H,Lassoc,Temps,Var),
19 %%   NewAcc = [Var|Acc],
20 %%   convertGeneListToVariableListAcc(T,Output,NewAcc).
21
22 % Convertit les expressors en liste de Taux d'influence
23 % on doit fournir le gene de dÃ©part
24 convertGeneListToRateList(Gene,Expressors,Lassoc,Temps,TauxList):-
25   %writeln(convertGeneListToRateList),
26   convertGeneListToRateListAcc(Gene,Expressors,Lassoc,Temps,TauxList,[]).
27
28 convertGeneListToRateListAcc(_,[],_,_,TauxList,TauxList).
29 convertGeneListToRateListAcc(Gene,[Hexpr|Texpr],Lassoc,Temps,TauxList,Acc):-
30   getRateFromActiveGene(Hexpr,Gene,Lassoc,Temps,Effet,_Niveau),
31 % nj(t) = nj(t-1) + w1j*n1(t-1) + w2j*n2(t-1) + w3j*n3(t-1) + ...
32   % "div 10" devrait etre sans doute "div BorneEffect"
33
34   %Tmp #= Effet * Niveau,
35   %Tmp2 #= Tmp div 10,
36   %NewAcc = [ Tmp2 |Acc],
37   NewAcc = [Effet | Acc],
38   convertGeneListToRateListAcc(Gene, Texpr,Lassoc,Temps,TauxList,NewAcc).
39
40
41
42
43
44 %/**************************************************************************/
45 % getRateFromActiveGene
46 %
47 %/**************************************************************************/
48
49
50 getRateFromActiveGene(GeneA,GeneB,Lassoc,_Temps,Effect,_NiveauPrev):-
51
52   integer(Effect), fd_size(Effect,1),
53   logger1("[getRateFromActiveGene] before_getRateFromActiveGene_cas_0"),logger1(GeneA),logger1
         ("->"), logger1(GeneB),logger1(":"), logger(Effect),
54   dif(GeneA,GeneB),
55   member(gen_gen(_,GeneA,GeneB, _,_,_Delay,Effect),Lassoc),
56   %Pas nÃ©cessaire ici -
57   % TempsPrev is Temps - Delay,
58   Effect = 0,
59   %!, %TODO
60   logger1("[getRateFromActiveGene] after_getRateFromActiveGene_cas_0"),logger1(GeneA),logger1(
         "->"), logger1(GeneB),logger1(":"), logger(Effect).
61
62 getRateFromActiveGene(GeneA,GeneB,Lassoc,Temps,EffectToConsider,NiveauPrev):-
63   logger1("[getRateFromActiveGene] cas_1 "),logger1(GeneA),logger1("->"), logger1(GeneB),
         logger1(":"), logger(Effect),
64   dif(GeneA,GeneB),
65   member(gen_gen(_,GeneA,GeneB,Threshold,_,Delay, Effect),Lassoc),
66   TempsPrev #= Temps - Delay - 1,
67   logger1("[getRateFromActiveGene] cas_1 TempsPrev =>"), logger(TempsPrev),
68   % TODO Quid si Delay n'est pas instantiÃ©
69   (getNiveau(GeneA,TempsPrev,NiveauPrev,Lassoc)-> logger1("[getRateFromActiveGene] cas_1 True
         NiveauPrev =>"), logger(NiveauPrev),
70                                 logger1("[getRateFromActiveGene] cas_1 True Threshold =>"),
                                       logger(Threshold),
71                                 NiveauPrev#<Threshold,
72                                 EffectToConsider #=0,
73                                 logger("[getRateFromActiveGene] END cas_1")
74
75   ;                             EffectToConsider #=0,
76                                 logger("[getRateFromActiveGene] END cas_1 False")),
77   logger1("[getRateFromActiveGene] cas_1"), logger1(after_getRateFromActiveGene_cas_0_),
         logger1(GeneA),logger1("->"), logger1(GeneB),logger1(":"), logger(Effect).
78
79 getRateFromActiveGene(GeneA,GeneB,Lassoc,Temps,EffectToConsider,NiveauPrev):-
80   logger1("[getRateFromActiveGene] cas_2 "),logger1(GeneA),logger1("->"), logger1(GeneB),
         logger1(":"), logger(Effect),
81   dif(GeneA,GeneB),
```

```
82    member(gen_gen(_,GeneA,GeneB,Threshold,_,Delay,Effect),Lassoc),
83    TempsPrev #= Temps - Delay -1 ,
84    logger1("[getRateFromActiveGene] cas_2 TempsPrev =>"), logger(TempsPrev),
85    %
86    (getNiveau(GeneA,TempsPrev,NiveauPrev,Lassoc) ->  logger1("[getRateFromActiveGene] cas_2
          True NiveauPrev =>"), logger1(niveauPrev_), logger(NiveauPrev),
87                                  logger1("[getRateFromActiveGene] cas_2 True Threshold =>"),
                                        logger1(threshold_), logger(Threshold),
88                                  NiveauPrev #>= Threshold,
89                                  EffectToConsider#=Effect,
90                                  logger("[getRateFromActiveGene] END cas_2 True")
91    ;                             EffectToConsider#=0,
92                                  logger("[getRateFromActiveGene] END cas_2 False ")),
93
94
95    logger1("[getRateFromActiveGene] cas_2"), logger1(after_getRateFromActiveGene_cas_1_),
          logger1(GeneA),logger1("->"), logger1(GeneB),logger1(":"), logger(Effect).
96
97
98  getRateFromActiveGene(Gene,Gene,Lassoc,Temps,Effect,NiveauPrev):-
99
100   TempsPrev #= Temps - Delay - 1,
101   %integer(Effect), fd_size(Effect,1),
102   logger1("[getRateFromActiveGene] cas_3 BEFORE ",logger1(GeneA),logger1("->"), logger1(GeneB
          ),logger1(":"), logger(Effect),
103   member(self(Gene,0),Lassoc),
104   Effect = 0,
105   %TODO!,
106   getNiveau(Gene,TempsPrev,NiveauPrev,Lassoc),
107   logger1("[getRateFromActiveGene] cas_3 AFTER ",logger1(GeneA),logger1("->"), logger1(GeneB)
          ,logger1(":"), logger(Effect).
108
109
110 getRateFromActiveGene(Gene,Gene,Lassoc,Temps,Effect,NiveauPrev):-
111
112   %TODO changer en Delay si nÃ©cessaire
113   TempsPrev #= Temps - Delay - 1,
114   %integer(Effect), fd_size(Effect,1),
115   logger1("[getRateFromActiveGene] cas_4 BEFORE"),logger1(GeneA),logger1("->"), logger1(GeneB)
          ,logger1(":"), logger(Effect),
116   member(self(Gene,Effect),Lassoc),
117   getNiveau(Gene,TempsPrev,NiveauPrev,Lassoc),
118
119   logger1("[getRateFromActiveGene] cas_4 AFTER "), logger1(niveauPrev_), logger(NiveauPrev),
120   logger1("[getRateFromActiveGene] cas_4 AFTER "), logger1(threshold_), logger(Threshold),
121   NiveauPrev #>= Threshold,
122   logger1("[getRateFromActiveGene] cas_4 AFTER"),logger1(GeneA),logger1("->"), logger1(GeneB),
          logger1(":"), logger(Effect).
123
124 getRateFromActiveGene(Gene,Gene,Lassoc,Temps,EffectToConsider,NiveauPrev):-
125   %TODO changer en Delay si nÃ©cessaire
126   TempsPrev #= Temps - Delay - 1,
127   logger1("[getRateFromActiveGene] cas_5 BEFORE"),logger1(GeneA),logger1("->"), logger1(GeneB)
          ,logger1(":"), logger(Effect),
128   member(self(Gene, Effect),Lassoc),
129   getNiveau(Gene,TempsPrev,NiveauPrev,Lassoc),
130   logger1("[getRateFromActiveGene] cas_5 self < threshold "), logger1(niveauPrev_), logger(
          NiveauPrev),
131   logger1("[getRateFromActiveGene] cas_5 self < threshold "), logger1(threshold_), logger(
          Threshold),
132   NiveauPrev#<Threshold,
133   EffectToConsider #=0,
134   logger1("[getRateFromActiveGene] cas_5"), logger1(after_getRateFromActiveGene_cas_0_),
          logger1(GeneA),logger1("->"), logger1(GeneB),logger1(":"), logger(Effect).


1   %but : contraindre une variable de niveau
2   contraindre_tmp([],_,_,_).
3   contraindre_tmp([self(_,_)|Tail],BorneMinNiveau,BorneMaxNiveau, Lassoc):-
4     %logger("[contraintre_tmp] self"),
5     contraindre_tmp(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc).
6   contraindre_tmp([gen_gen(_,_,_,_,_,_,_)|Tail],BorneMinNiveau,BorneMaxNiveau, Lassoc):-
7     contraindre_tmp(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc).
8
9   contraindre_tmp([niveau(control,_,_,_)|Tail],BorneMinNiveau,BorneMaxNiveau, Lassoc):-
10    logger("[contraintre_tmp] niveau control => not constrained"),
11    contraindre_tmp(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc).
```

```
12
13 contraindre_tmp([niveau(node,_,0,_)|Tail],BorneMinNiveau,BorneMaxNiveau, Lassoc):-
14   contraindre_tmp(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc).
15
16 contraindre_tmp([niveau(node,Gene,Temps,NiveauAContraindre)|Tail],BorneMinNiveau,
       BorneMaxNiveau, Lassoc):-
17   Temps > 0,
18   logger("[contraintre_tmp] BEGIN"),
19   logger1("[contraintre_tmp] niveau, temps > 0 => "),logger(Temps),
20   logger1("[contraintre_tmp] Gene => "), logger(Gene),
21   TempsPrev is Temps - 1,
22
23   findall(GeneFA, member(gen_gen(_,GeneFA,Gene,_,_,_,_), Lassoc), Expressors),
24   % From L
25   getAllInfo(Expressors,Gene,Temps,Lassoc,NiveauFAs,ThresholdFAs,EffectFAs),
26   apply_predicate_list(NiveauFAs,ThresholdFAs,Integer_GT),
27   scalar_product(Integer_GT,EffectFAs,#=,Contribution),
28
29   logger1("[contraintre_tmp] Niveau => "), logger(NiveauFAs),
30   logger1("[contraintre_tmp] Threshold => "), logger(ThresholdFAs),
31   logger1("[contraintre_tmp] Integer_GT => "), logger(Integer_GT),
32
33   getNiveau(Gene,TempsPrev,NiveauPrev,Lassoc),
34   NiveauTmp#=NiveauPrev + Contribution,
35   boundaries(NiveauTmp,NiveauAContraindre,BorneMinNiveau,BorneMaxNiveau),
36   logger("[contraintre_tmp] END"),
37   contraindre_tmp(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc).
38
39 %
40 % permet de couper la liste MegaList (une liste de listes), en trois listes:
41 % MegaList = [[A1,A2,A3],[B1,B2,B3],[C1,C2,C3]]
42 % L1 = [A1, B1, C1]
43 % L2 = [A2, B2, C2]
44 % L3 = [A3, B3, C3]
45
46 split_expr_thresh_eff(ListeIn,L1,L2,L3):-
47   split_expr_thresh_eff_(ListeIn,L1, [], L2, [], L3,[]).
48
49 split_expr_thresh_eff_([],L1, L1, L2, L2, L3,L3).
50 split_expr_thresh_eff_([[H1,H2,H3]|T],L1, AccExpr, L2, AccThresh, L3,AccEffects):-
51   split_expr_thresh_eff_(T,L1,[H1|AccExpr], L2, [H2|AccThresh], L3,[H3|AccEffects]).
52
53
54
55 % getAllInfo
56 % From Expressors, GeneCible, TempsPrev et Lassoc,
57 % Fournit la liste des niveaux des genes (dans l'ordre), des thresholds, et des effects
58 % des expresseurs
59
60 getAllInfo(Expressors,GeneCible,Temps,Lassoc,NiveauFAs,ThresholdFAs,EffectFAs):-
61   %logger1("[getAllInfo] ListeGene -> "), logger(ListeGene),
62   %logger1("[getAllInfo] TempsPrev -> "), logger(TempsPrev),
63   getAllInfoMegaList(Expressors,GeneCible,Temps,Lassoc,MegaList),!,
64   split_expr_thresh_eff(MegaList,NiveauFAs,ThresholdFAs,EffectFAs).
65
66
67 getAllInfoMegaList(Expressors,GeneCible,Temps,Lassoc,MegaList):-
68   getAllInfoMegaList_(Expressors,GeneCible,Temps,Lassoc,MegaList,[]).
69
70 % Recursif sur la liste des expresseurs
71 getAllInfoMegaList_([],_,_,_,MegaList,MegaList).
72 getAllInfoMegaList_([HExpressors|TExpressors],GeneCible,Temps,Lassoc,MegaList,Acc):-
73   getAllInfoRecLassoc(HExpressors, GeneCible, Temps, Lassoc, MegaListPart, [], Lassoc),
74   append(MegaListPart, Acc, NewAcc),
75   getAllInfoMegaList_(TExpressors,GeneCible,Temps,Lassoc,MegaList,NewAcc).
76
77 % Recursif sur la liste d'association directement
78 getAllInfoRecLassoc(HExpressors,GeneCible,_, Lassoc, MegaListPart, MegaListPart, _):-
79   not(member(gen_gen(_,HExpressors,GeneCible,_,_,_,_), Lassoc)).
80
81 getAllInfoRecLassoc(GeneFA,GeneCible, Temps, Lassoc, MegaListPart, AccPart, L):-
82   logger("[getAllInfoRecLassoc] ----------begin----------"),
83   select(gen_gen(_,GeneFA,GeneCible,ThresholdFA, _,DelayFA, EffectFA),Lassoc,LassocRest),
84   logger("[getAllInfoRecLassoc] after select DelayFA"),
85   TempsPrev #= Temps - DelayFA - 1,
86   logger("[getAllInfoRecLassoc] TempsPrev => "), logger(TempsPrev),
```

```
87
88    (select(niveau(_,GeneFA,TempsPrev,NiveauFA),LassocRest,LassocRest2)
89    ->          true,
90                logger("[getAllInfoRecLassoc] True"),
91                AccTmp = [NiveauFA,ThresholdFA,EffectFA],
92                getAllInfoRecLassoc(GeneFA,GeneCible,Temps,LassocRest2, MegaListPart, [AccTmp|
                      AccPart], L)
93    ;           logger("[getAllInfoRecLassoc] False"),
94                getAllInfoRecLassoc(GeneFA,GeneCible,Temps,LassocRest, MegaListPart, AccPart, L)),
95    logger("[getAllInfoRecLassoc] ----------------end----------------")
96    .
97
98
99  % Ls3 est la liste des 1 ou 0 en fonction de si les Ã©lÃ©ments de L1 sont (ou non) plus grands
        que les Ã©lements de L2.
100 apply_predicate_list(Ls1,Ls2,Ls3):-
101   maplist(gt_threshold,Ls1, Ls2, Ls3).
102
103 gt_threshold(L1,L2,L3) :-
104   L1 #>= L2,
105   L3 is 1.
106 gt_threshold(L1,L2,L3) :-
107   L1 #< L2,
108   L3 is 0.

 1  contraindre_memoisation(Lassoc,BorneMinNiveau,BorneMaxNiveau):-
 2
 3      fixer_delay(0,Lassoc),
 4      msort(Lassoc,LassocSorted),
 5      logger1("[contraindre_memoisation] Lassoc = "), logger(Lassoc),
 6      contraindre_memoisation_main(LassocSorted, BorneMinNiveau, BorneMaxNiveau, LassocSorted,
            Structure),
 7      apply_constraint(BorneMinNiveau,BorneMaxNiveau,Structure).
 8
 9
10  contraindre_memoisation_main(Lassoc,BLow,BUp, LassocBase, Out):-
11      contraindre_memoisation_(Lassoc,BLow,BUp, LassocBase,Out,[]).
12
13  contraindre_memoisation_([],_,_,_,Out,Out).
14  contraindre_memoisation_([gen_gen(_,_,_,_,_,_,_,_)|Tail],BorneMinNiveau,BorneMaxNiveau,Lassoc,
        Out,Acc):-
15    contraindre_memoisation_(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc,Out,Acc).
16  contraindre_memoisation_([self(_,_)|Tail],BorneMinNiveau,BorneMaxNiveau, Lassoc,Out,Acc):-
17    contraindre_memoisation_(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc,Out,Acc).
18
19  contraindre_memoisation_([niveau(control,_,_,_)|Tail], BorneMinNiveau,BorneMaxNiveau, Lassoc,
        Out,Acc):-
20    logger("[contraindre_memoisation_] niveau control => not constrained"),
21    contraindre_memoisation_(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc,Out,Acc).
22
23  contraindre_memoisation_([niveau(node,Gene,0,_)|Tail], BorneMinNiveau,BorneMaxNiveau, Lassoc,
        Out,Acc):-
24    member(niveau(_,Gene,1,_), Lassoc),
25    contraindre_memoisation_(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc,Out,Acc).
26
27  contraindre_memoisation_([niveau(node,Gene,Temps,Var)|Tail],BorneMinNiveau,BorneMaxNiveau,
        Lassoc,Out,Acc):-
28
29    Temps>0,
30    TempsPrev is Temps - 1,
31    logger1("[contraindre_memoisation_] -Gene,Temps :"), logger1(Gene),logger1(", "), logger(
          Temps),
32    getNiveau(Gene,TempsPrev,VarPrev,Lassoc),
33    % recuperer les thresholds/Effects des associations et Niveaux associÃ©s
34    getBagThresholdsEffectsNiveaux(TempsPrev, Gene, Lassoc,BagThresholds,BagEffects,
          _BagPairsGeneNiveau,BagNiveaux),
35
36    % Les mettre dans le meme ordre
37    keysort(BagThresholds,PairThresholds),
38    keysort(BagEffects,PairEffects),
39    keysort(BagNiveaux,PairNiveaux),
40
41    % Ne garder que les valeurs (on n a pas besoins de conserver les cles = geneA, geneB, ...)

42    pairs_values(PairThresholds, Thresholds),
43    pairs_values(PairEffects, Effects),
```

```
44    pairs_values(PairNiveaux, Niveaux),
45
46    NewAcc=[[Var,VarPrev]-[Niveaux,Thresholds,Effects]|Acc],
47    contraindre_memoisation_(Tail,BorneMinNiveau,BorneMaxNiveau,Lassoc,Out,NewAcc).
48
49
50  apply_constraint(_,_,[]):-
51    logger("[apply_constraint] End").
52  apply_constraint(BorneMinNiveau,BorneMaxNiveau,Matrix):-
53    length(Matrix,LengthMatrix),
54    logger1("[apply_constraint] Main, MATRIX Length = ",logger(LengthMatrix),
55
56    Matrix = [[Var,VarPrev]-[Niveaux,Thresholds,Effects]|Tail],
57
58    maplist(myzcompare,Relations,Niveaux,Thresholds,TrueVal),
59    scalar_product(TrueVal, Effects, #=, Contribution),
60
61    VarTmp #= VarPrev+Contribution,
62    boundaries(VarTmp,Var,BorneMinNiveau,BorneMaxNiveau),
63    apply_constraint(BorneMinNiveau,BorneMaxNiveau,Tail).
64
65
66
67  getBagThresholdsEffectsNiveaux(TempsPrev, Gene, Lassoc,BagThresholds,BagEffects,
        BagPairsGeneNiveau,BagNiveaux):-
68    getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, Lassoc,BagThresholds,BagEffects,
          BagPairsGeneNiveau,[],[],[]),
69    getPairsGeneNiveau(BagEffects,BagPairsGeneNiveau,BagNiveaux).
70
71
72  % cas de base : liste vide
73  getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, [],BagThresholds,BagEffects,BagPairsGeneNiveau
        ,BagThresholds,BagEffects,BagPairsGeneNiveau).
74
75  % cas self -> rien a faire
76  getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, [self(_,_)| TailLassoc],BagThresholds,
        BagEffects,BagPairsGeneNiveau,AccThresholds,AccEffects,AccPairsGeneNiveau):-
77    getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, TailLassoc,BagThresholds,BagEffects,
          BagPairsGeneNiveau, AccThresholds,AccEffects, AccPairsGeneNiveau).
78
79  % cas "niveau" ou le temps = tempsPrev ==> on ajoute dans l'accumulateur
80  getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, [niveau(_,GeneTmp,TempsPrev,NiveauTmp)|
        TailLassoc],BagThresholds,BagEffects,BagPairsGeneNiveau,AccThresholds,AccEffects,
        AccPairsGeneNiveau):-
81    getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, TailLassoc,BagThresholds,BagEffects,
          BagPairsGeneNiveau,AccThresholds,AccEffects,[GeneTmp-NiveauTmp | AccPairsGeneNiveau]).
82
83  % cas "niveau" ou le temps =/= tempsPrev ==> on ne fait rien
84  getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, [niveau(_,_,Temps,_)| TailLassoc],
        BagThresholds,BagEffects,BagPairsGeneNiveau,AccThresholds,AccEffects,AccPairsGeneNiveau):-
85    dif(TempsPrev,Temps),
86    getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, TailLassoc,BagThresholds,BagEffects,
          BagPairsGeneNiveau,AccThresholds,AccEffects,AccPairsGeneNiveau).
87
88
89  %% cas "gen_gen" ou le geneTo = gene ==> on ajoute dans l'accumulateur
90  getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, [gen_gen(_,From,Gene,Threshold,_,_,Effect)|
        TailLassoc], BagThresholds, BagEffects, BagPairsGeneNiveau, AccThresholds, AccEffects,
        AccPairsGeneNiveau):-
91    getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, TailLassoc,BagThresholds,BagEffects,
          BagPairsGeneNiveau,[From-Threshold | AccThresholds],[From-Effect | AccEffects],
          AccPairsGeneNiveau).
92  %% cas "gen_gen" ou le geneTo =/= gene ==> on ne fait rien
93  getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, [gen_gen(_,_From,GeneTo,_Threshold,_,_,_Effect
        )| TailLassoc], BagThresholds, BagEffects, BagPairsGeneNiveau, AccThresholds, AccEffects,
        AccPairsGeneNiveau):-
94    dif(Gene,GeneTo),
95    getBagThresholdsEffectsNiveaux_(TempsPrev,Gene, TailLassoc,BagThresholds,BagEffects,
          BagPairsGeneNiveau,AccThresholds,AccEffects,AccPairsGeneNiveau).
96
97  %getPairsGeneNiveau(BagEffects,BagPairsGeneNiveau,BagNiveaux):-
98  getPairsGeneNiveau(BagEffects,BagPairsGeneNiveau,BagNiveaux):-
99    getPairsGeneNiveau_(BagEffects,BagPairsGeneNiveau,BagNiveaux,[]).
100
101 getPairsGeneNiveau_([],_,BagNiveaux,BagNiveaux).
102 getPairsGeneNiveau_([GeneTmp-_ | TailBagEffects],BagPairsGeneNiveau,BagNiveaux,Acc):-
```

```
103     member(GeneTmp-NiveauTmp, BagPairsGeneNiveau),
104     getPairsGeneNiveau_(TailBagEffects,BagPairsGeneNiveau,BagNiveaux,[GeneTmp-NiveauTmp|Acc]).
105
106   getPairsGeneNiveau_([GeneTmp-_ | TailBagEffects],BagPairsGeneNiveau,BagNiveaux,Acc):-
107     %logger1("[DEBUG] KO GeneTmp-G1 => "), logger(GeneTmp-_),
108     %logger1("[DEBUG] KO BagPairsGeneNiveau"), logger(BagPairsGeneNiveau),
109     logger("[getPairsGeneNiveau] SHOULD NEVER BE HERE"),
110     not(member(GeneTmp-NiveauTmp, BagPairsGeneNiveau)),
111     logger("[getPairsGeneNiveau] Will Fail"),
112     fail.
113
114
115   % Le cas ou le niveau=0 est necessaire
116   % Gestion d'un rÃ©seau ou les dÃ©lais font qu'on n'a aucune contributiion plus tard
117
118   myzcompare(Rel,N1,T1,Val):-
119     zcompare(Rel,N1,T1),
120     ord_(Rel,Val).
121   ord_(>,1).
122   ord_(<,0).
123   ord_(=,1).


  1   % CONTRAINDRE_ASYNCHRONE
  2   % DOIT COMMENCER AVEC TEMPS = (N)
  3
  4   contraindre_asynchrone_main(N,BorneMinNiveau,BorneMaxNiveau,Lassoc):-
  5     logger("[contraintre_asynchrone_main] BEGIN"),
  6     getDelays(Lassoc,Delays),
  7     length(Delays,LengthDelays),
  8     Up #= LengthDelays-1,
  9     Delays ins 0..Up,
 10     logger("[contraintre_asynchrone_main] Before Rec"),
 11     all_distinct(Delays),
 12     contraindre_asynchrone(N,BorneMinNiveau,BorneMaxNiveau,LengthDelays,Lassoc).
 13
 14
 15   contraindre_asynchrone(0, _,_,_, _):- %TODO
 16     logger("[contraindre_asynchrone] Temps -> 0"),
 17     logger(contraindre_niveaux_0).
 18
 19   contraindre_asynchrone(Temps,BorneMinNiveau,BorneMaxNiveau, LengthDelays, Lassoc):-
 20     logger("[contraindre_asynchrone]----------------BEGIN----------------"),
 21     Temps>0,
 22     TempsPrev is Temps - 1,
 23
 24     %Find Expressors
 25     getRequiredExpressor(Temps, GeneFrom,GeneTo,LengthDelays, Lassoc),
 26
 27     logger1("[contraindre_asynchrone] Temps -> "),logger(Temps),
 28     logger1("[contraindre_asynchrone] after getRequiredExpressor GeneFrom -> "),logger(GeneFrom)
          ,
 29     logger1("[contraindre_asynchrone] after getRequiredExpressor GeneTo -> "),logger(GeneFrom),
 30
 31     getEffectFromActiveGene(GeneFrom,GeneTo,Lassoc,Temps,EffectToConsider,_Niveau),
 32
 33     logger1("[contraindre_asynchrone] EffectToConsider -> "),logger(EffectToConsider),
 34
 35     getNiveau(GeneTo,TempsPrev,VarPrev,Lassoc),
 36     getNiveau(GeneTo,Temps,Var,Lassoc),
 37     logger1("[contraindre_asynchrone] Var -> "),logger(Var),
 38     logger1("[contraindre_asynchrone] VarPrev -> "),logger(VarPrev),
 39
 40     VarTmp #= VarPrev + EffectToConsider,
 41     boundaries(VarTmp,Var,BorneMinNiveau,BorneMaxNiveau),
 42
 43     logger1("[contraindre_asynchrone] GeneTo-Var -> "),logger1(GeneTo),logger1(" - "), logger(
          Var),
 44     logger("[contraindre_asynchrone] before remaining"),
 45     contraindre_asynchrone_remaining(Temps,GeneTo,Lassoc,Lassoc),
 46     logger("[contraindre_asynchrone] after remaining"),
 47     logger("[contraindre_asynchrone]----------------END----------------"),
 48     contraindre_asynchrone(TempsPrev,BorneMinNiveau,BorneMaxNiveau,LengthDelays,Lassoc).
 49
 50
 51   % Donner la meme valeur qu'au temps prÃ©cÃ©dent
 52   contraindre_asynchrone_remaining(_,_,[],_):-
```

```
53    logger("[contraindre_asynchrone_remaining] end of recursion").
54
55  % cas ou GeneTo =/= Gene, mais les temps sont ok
56  contraindre_asynchrone_remaining(Time,GeneTo,[niveau(node,Gene,Time,Var)|TBag],Lassoc):-
57    %logger1("[contraindre_asynchrone_remaining] BEGIN -> "), logger(niveau(node,Gene,Time,Var))
          ,
58    dif(Gene,GeneTo),
59    TimePrev #= Time -1,
60    getNiveau(Gene,TimePrev,VarPrev,Lassoc),
61    Var #= VarPrev,
62    contraindre_asynchrone_remaining(Time,GeneTo,TBag,Lassoc).
63  contraindre_asynchrone_remaining(Time,GeneTo,[niveau(node,GeneTo,_,Var)|TBag],Lassoc):-
64    %logger1("[contraindre_asynchrone_remaining] BEGIN -> "), logger(niveau(node,Gene,Time,Var))
          ,
65    contraindre_asynchrone_remaining(Time,GeneTo,TBag,Lassoc).
66
67  % cas ou GeneTo =/= Gene, mais les temps sont =/=
68  contraindre_asynchrone_remaining(Time,GeneTo,[niveau(node,GeneA,TimeA,_)|TBag],Lassoc):-
69    %logger1("[contraindre_asynchrone_remaining] BEGIN -> "), logger(niveau(node,Gene,Time,Var))
          ,
70    dif(GeneTo,GeneA),
71    dif(Time,TimeA),
72    contraindre_asynchrone_remaining(Time,GeneTo,TBag,Lassoc).
73
74  contraindre_asynchrone_remaining(Time,GeneTo,[niveau(control,_,_,_)|TBag],Lassoc):-
75    contraindre_asynchrone_remaining(Time,GeneTo,TBag,Lassoc).
76  contraindre_asynchrone_remaining(Time,GeneTo,[self(_,_)|TBag],Lassoc):-
77    contraindre_asynchrone_remaining(Time,GeneTo,TBag,Lassoc).
78  contraindre_asynchrone_remaining(Time,GeneTo,[gen_gen(_,_,_,_,_,_,_)|TBag],Lassoc):-
79    contraindre_asynchrone_remaining(Time,GeneTo,TBag,Lassoc).
80
81
82  % To get all the Delays variables out of Lassoc
83  getDelays(Lassoc,Delays):-
84    getDelays_(Lassoc,Delays,[]).
85
86  getDelays_([],Delays,Delays).
87  getDelays_([gen_gen(_,_,_,_,_,Delay,_)|Tail],Delays,Acc):-
88      getDelays_(Tail,Delays,[Delay|Acc]).
89  getDelays_([niveau(_,_,_,_)|Tail],Delays,Acc):-
90      getDelays_(Tail,Delays,Acc).
91  getDelays_([self(_,_)|Tail], Delays,Acc):-
92      getDelays_(Tail,Delays,Acc).
93
94
95  % To get the required expressor, in a cyclic way
96  getRequiredExpressor(Temps, GeneFrom,GeneTo,LengthDelays, Lassoc):-
97    member(gen_gen(_,GeneFrom,GeneTo,_,_,Delay,_), Lassoc),
98    Delay #= Temps mod LengthDelays,
99    logger1("[getRequiredExpressor] Temps => "), logger(Temps),
100   logger1("[getRequiredExpressor] Delay => "), logger(Delay),
101   logger1("[getRequiredExpressor] LengthDelays => "), logger(LengthDelays).
102
103 getEffectFromActiveGene(GeneA,GeneB,Lassoc,_Temps,Effect,_NiveauPrev):-
104   integer(Effect), fd_size(Effect,1),
105   logger1("[getRateFromActiveGene] before_getRateFromActiveGene_cas_0"),logger1(GeneA),logger1
          ("->"), logger1(GeneB),logger1(":"), logger(Effect),
106   dif(GeneA,GeneB),
107   member(gen_gen(_,GeneA,GeneB, _,_,_Delay,Effect),Lassoc),
108   %Pas nÃ©cessaire ici -
109   % TempsPrev is Temps - Delay,
110   Effect = 0,
111   logger1("[getRateFromActiveGene] after_getRateFromActiveGene_cas_0"),logger1(GeneA),logger1(
          "->"), logger1(GeneB),logger1(":"), logger(Effect).
112
113 getEffectFromActiveGene(GeneA,GeneB,Lassoc,Temps,EffectToConsider,NiveauPrev):-
114   logger1("[getEffectFromActiveGene] cas_1 "),logger1(GeneA),logger1("->"), logger1(GeneB),
          logger1(":"), logger(Effect),
115   dif(GeneA,GeneB),
116   member(gen_gen(_,GeneA,GeneB,Threshold,_,_, Effect),Lassoc),
117   getNiveau(GeneA,TempsPrev,NiveauPrev,Lassoc),
118   TempsPrev #= Temps - 1, %TODO : Pas de Delay ici
119   logger1("[getEffectFromActiveGene] cas_1 TempsPrev => "), logger(TempsPrev),
120   logger1("[getEffectFromActiveGene] cas_1 True NiveauPrev => "), logger(NiveauPrev),
121   logger1("[getEffectFromActiveGene] cas_1 True Threshold => "), logger(Threshold),
122   NiveauPrev#<Threshold,
```

```
123     EffectToConsider #=0,
124     logger("[getEffectFromActiveGene] END cas_1"),
125     logger1("[getEffectFromActiveGene] cas_1"), logger1(after_getRateFromActiveGene_cas_0_),
            logger1(GeneA),logger1("->"), logger1(GeneB),logger1(":"), logger(Effect).
126
127   getEffectFromActiveGene(GeneA,GeneB,Lassoc,Temps,EffectToConsider,NiveauPrev):-
128     logger1("[getEffectFromActiveGene] cas_2 "),logger1(GeneA),logger1("->"), logger1(GeneB),
            logger1(":"), logger(Effect),
129     dif(GeneA,GeneB),
130     member(gen_gen(_,GeneA,GeneB,Threshold,_,_,EffectToConsider),Lassoc),
131     TempsPrev #= Temps -1 ,
132     logger1("[getEffectFromActiveGene] cas_2 TempsPrev => "), logger(TempsPrev),
133     getNiveau(GeneA,TempsPrev,NiveauPrev,Lassoc),
134     logger1("[getEffectFromActiveGene] cas_2 True NiveauPrev => "), logger1(niveauPrev_), logger
            (NiveauPrev),
135     logger1("[getEffectFromActiveGene] cas_2 True Threshold => "), logger1(threshold_), logger(
            Threshold),
136     NiveauPrev #>= Threshold,
137     logger("[getEffectFromActiveGene] END cas_2 True"),
138     logger1("[getEffectFromActiveGene] cas_2"), logger1(after_getRateFromActiveGene_cas_1_),
            logger1(GeneA),logger1("->"), logger1(GeneB),logger1(":"), logger(Effect).
```

A.4.3 Sparsity

```
1   % sparsity : Parmi tous les EFFETS, au moins "Coef" ont une valeur >= a un niveau,
2   % et au moins Coef ont une valeur <= threshold
3   %-------------------------------------------------------------------------%
4   % But : limiter le nombre d'interactions Ã  un nombre donnÃ©e             %
5   %-------------------------------------------------------------------------%
6
7   sparsity(0,_,_):-!.
8   sparsity(NbreMaxEffect,BorneEffectOnOthers,Lassoc):-
9
10    % On recupere toutes les variables "Effect"
11    bagof(Var, A^B^C^D^E^F^(member(gen_gen(A,B,C,D,E,F,Var),Lassoc)),BagOfVar),
12    % On optient leur nombre => parmi celles lÃ , seules NbreMaxEffect peuvent Ãªtre diffÃ©rents
          de 0
13    length(BagOfVar, Len_Effects_Potentiels),
14
15    NbreMinZero is Len_Effects_Potentiels-NbreMaxEffect,
16    logger1("[sparsity] Liste Effects ->"),logger(BagOfVar),
17
18    Low is -BorneEffectOnOthers,
19    Up is BorneEffectOnOthers,
20
21    % On crÃ©e la liste [-BorneEffect,BorneEffect] et on rÃ©cupÃ¨re sa version sans 0.
22    numlist(Low,Up,List),
23    select(0,List,ListSansZero),
24
25  % ?- pairs_keys_values(Pairs,[1,2,3],Inc).
26  %    Pairs = [1-A, 2-B, 3-C],
27  %    Inc = [A, B, C].
28
29    pairs_keys_values(Pairs,ListSansZero,Inc),
30    logger1("[sparsity] Inc ->"),logger(Inc),
31    % Inc contient la liste des nombres d'Effects qui sont diffÃ©rents de 0
32    Inc ins 0..NbreMaxEffect,
33    sum(Inc,#=<,NbreMaxEffect),
34
35    MaxReal #>=NbreMinZero,
36
37    append([0-MaxReal],Pairs,NewPairs),
38    logger1("[sparsity] Liste Effects ->"),logger(BagOfVar),
39    logger1("[sparsity] Pairs Key-Num ->"),logger(NewPairs),
40
41    global_cardinality(BagOfVar,NewPairs),
42    logger1("[sparsity] Fin de Sparsity").
```

A.4.4 Labeling

```
1   %-------------------------------------------------------------------------%
2   %                                                                         %
3   %                              LABELING                                   %
4   %                                                                         %
```

```
5  % ------------------------------------------------------------------------ %
6
7  % Labeling: options
8  % none = label_all (incompatible avec les autres)
9  % minimize_effects = minimizer la somme de tous les effects
10 % min_threshold = Limiter les thresholds a la borne infÃ©rieure
11 % max_borne = limiter l'effect Ã  la borne max
12 % niveaux_only =
13 %
14 my_labeling(Lassoc):-
15     label_all(ff, Lassoc).
16 my_labeling(optimized,Lassoc):-
17   label_niveaux(Lassoc),
18     label_effect(Lassoc),
19     label_borneEffect(Lassoc),!,
20     label_threshold(Lassoc),!,
21     label_delay(Lassoc),!.
22 my_labeling(all_ok,Lassoc):-
23   label_all(ff, Lassoc).
24 my_labeling(all_ff,Lassoc):-
25   label_all(ff, Lassoc).
26
27
28 % ------------------------------------------------------------------------ %
29 %                                                                          %
30 %                                NIVEAUX                                   %
31 %                                                                          %
32 % ------------------------------------------------------------------------ %
33 label_niveaux(Lassoc) :-
34   logger("Debut␣Labeling␣Niveau"),
35   label_niveaux_aux(Lassoc,[]),
36   logger("Fin␣labeling␣Niveau"),
37   logger(Lassoc).
38
39 label_niveaux_aux([],L):-
40   labeling([ffc],L).
41 label_niveaux_aux([niveau(_,_,_,Niveau)|Tail],L):-
42   label_niveaux_aux(Tail,[Niveau|L]).
43 label_niveaux_aux([gen_gen(_,_,_,_,_,_,_)|Tail],L):-
44   label_niveaux_aux(Tail,L).
45 label_niveaux_aux([self(_,_)|Tail],L):-
46   label_niveaux_aux(Tail,L).
47
48
49 % ------------------------------------------------------------------------ %
50 %                                                                          %
51 %                                EFFECTS                                   %
52 %                                                                          %
53 % ------------------------------------------------------------------------ %
54 label_effect(Lassoc) :-
55   logger("Debut␣Labeling␣Effet"),
56   label_effect_aux(Lassoc,[]),
57   logger("Fin␣labeling␣Effet"),
58   logger(Lassoc),
59   logger("Prune␣Effect0"),
60   prune_effect0(Lassoc).
61
62 %label_effect_aux([],L):-
63 % write(L),
64 % maplist(fd_dom, L,DomL),
65 % write(DomL),
66 % maplist(dom_integers,DomL,ListDom),
67 % write(ListDom),
68 % maplist(#=,L,ListDom),
69 % write(L).
70 % %labeling([ff],L).
71
72
73 % This is already an improvement : first solutions delivered are based on Effect = 0
74 label_effect_aux([],L):-
75   valeur_absolue(L,Labs),
76   sum(Labs,#=,Sum),
77   logger1("[label_effect]␣SUM␣=>␣"), logger(Sum),
78   labeling([min(Sum),ffc,bisect],L).
79
80 label_effect_aux([niveau(_,_,_,_)|Tail],L):-
```

```
81    label_effect_aux(Tail,L).
82  label_effect_aux([gen_gen(_,_,_,_,_,_,Effect)|Tail],L):-
83    label_effect_aux(Tail,[Effect|L]).
84  label_effect_aux([self(_,Effect)|Tail],L):-
85    label_effect_aux(Tail,[Effect|L]).
86
87
88  %-------------------------------------------------------------------------%
89  %                                                                         %
90  %                              BORNE EFFECTS                              %
91  %                                                                         %
92  %-------------------------------------------------------------------------%
93
94  % This is already an improvement : BorneEffect are majoring their interval
95  label_borneEffect(Lassoc) :-
96    logger("Labeling BorneEffect"),
97    label_borneEffect_aux(Lassoc,[]),
98    logger("End Labeling BorneEffect"),
99    logger(Lassoc).
100
101 label_borneEffect_aux([],L):-
102   maplist(fd_sup, L,DomL),
103   maplist(#=,L,DomL),
104   labeling([ff],L).
105 label_borneEffect_aux([niveau(_,_,_,_)|Tail],L):-
106   label_borneEffect_aux(Tail,L).
107 label_borneEffect_aux([self(_,_)|Tail], L):-
108   label_borneEffect_aux(Tail,L).
109 label_borneEffect_aux([gen_gen(_,_,_,_,Borne,_,_)|Tail], L):-
110   label_borneEffect_aux(Tail,[Borne|L]).
111
112
113
114 %-------------------------------------------------------------------------%
115 %                                                                         %
116 %                              THRESHOLDS                                 %
117 %                                                                         %
118 %-------------------------------------------------------------------------%
119 % This is already an improvement : Threshold are minoring their interval
120 label_threshold(Lassoc) :-
121   logger("Labeling Threshold"),
122   label_threshold_aux(Lassoc,[]),
123   logger("End Labeling Threshold"),
124   logger(Lassoc).
125
126 label_threshold_aux([],L):-
127   maplist(fd_inf, L,DomL),
128   maplist(#=,L,DomL),
129   logger1("DOMAIN RESTANT THRESHOLDS = "), logger(DomL),
130   labeling([ff],L).
131 label_threshold_aux([niveau(_,_,_,_)|Tail],L):-
132   label_threshold_aux(Tail,L).
133 label_threshold_aux([self(_,_)|Tail], L):-
134   label_threshold_aux(Tail,L).
135 label_threshold_aux([gen_gen(_,_,_,Threshold,_,_,_)|Tail], L):-
136   label_threshold_aux(Tail,[Threshold|L]).
137
138
139
140 %-------------------------------------------------------------------------%
141 %                                                                         %
142 %                                DELAY                                    %
143 %                                                                         %
144 %-------------------------------------------------------------------------%
145 label_delay(Lassoc) :-
146   logger("Labeling Threshold"),
147   label_delay_aux(Lassoc,[]),
148   logger("End Labeling Threshold"),
149   logger(Lassoc).
150
151 label_delay_aux([],L):-
152   %maplist(fd_inf, L,DomL),
153   %maplist(#=,L,DomL),!,
154   labeling([ff],L).
155 label_delay_aux([niveau(_,_,_,_)|Tail],L):-
156   label_delay_aux(Tail,L).
```

```
157 label_delay_aux([self(_,_)|Tail], L):-
158   label_delay_aux(Tail,L).
159 label_delay_aux([gen_gen(_,_,_,_,_,Delay,_)|Tail], L):-
160   label_delay_aux(Tail,[Delay|L]).
161
162
163
164 %-------------------------------------------------------------------------%
165 %                                                                         %
166 %                                 ALL                                     %
167 %                                                                         %
168 %-------------------------------------------------------------------------%
169
170 % On labelise tout en mÃªme temps
171 label_all(Option,Lassoc) :-
172     logger("Labeling ALL"),
173     label_all_aux(Option,Lassoc,[]),
174     logger("Labeling ALL END"),
175     logger(Lassoc).
176
177 label_all_aux(ff,[],L):-
178   labeling([ff],L).
179 label_all_aux(std,[],L):-
180   label(L).
181 label_all_aux(Option,[niveau(node,_,_,Place)|Tail],L):-
182   label_all_aux(Option,Tail,[Place|L]).
183 label_all_aux(Option,[niveau(control,_,_,Place)|Tail],L):-
184   label_all_aux(Option,Tail,[Place|L]).
185 label_all_aux(Option,[gen_gen(_,_,_,Threshold,BorneEffect,Delay,Effect)|Tail],L):-
186   append([Threshold,BorneEffect,Delay,Effect], L, Lnew),
187   label_all_aux(Option,Tail,Lnew).
188 label_all_aux(Option,[self(_,Effect)|Tail],L):-
189   label_all_aux(Option,Tail,[Effect|L]).
190
191
192
193
194 %http://www.pathwayslms.com/swipltuts/clpfd/clpfd.html#_optimization
195 dom_integers(D, Is) :- phrase(dom_integers_(D), Is).
196
197 dom_integers_(I)      --> { integer(I) }, [I].
198 dom_integers_(L..U)   --> { numlist(L, U, Is) }, Is.
199 dom_integers_(D1\/D2) --> dom_integers_(D1), dom_integers_(D2).
200
201
202
203
204
205
206 % Si Effect est deja assignÃ© Ã  0, force le Threshold et la BorneEffect Ã  valeur min du
      domaine.
207 % Delay n'est pas forcÃ© pour compatibilitÃ© avec asynchronous
208 prune_effect0([]).
209 prune_effect0([gen_gen(_,_,_,Threshold,BorneEffect,Delay,Effect)|Tail]):-
210   integer(Effect),
211   Effect #= 0,
212   %write("Threshold -> "),
213   fd_inf(Threshold,MinThreshold), Threshold#=MinThreshold,
214   %write("Borne -> "),
215   fd_inf(BorneEffect,MinBorneEffect), BorneEffect#=MinBorneEffect,
216   %write("Delay -> "),
217   %fd_inf(Delay,MinDelay), Delay#=MinDelay,
218   prune_effect0(Tail).
219 prune_effect0([gen_gen(_,_,_,_,_,_,_)|Tail]):-
220   prune_effect0(Tail).
221 prune_effect0([niveau(_,_,_,_)|Tail]):-
222   prune_effect0(Tail).
223 prune_effect0([self(_,_)|Tail]):-
224   prune_tree_effect0(Tail).
```

A.5 Logger

```
1 %createLog.
2 createLog :-
```

```
3     get_time(X),
4     truncate(X,2,Y),
5     atomic_concat(Y,".txt",FileName),
6     atomic_concat("logs/",FileName,CompleteName),
7     open(CompleteName,append,MS,[alias(stream)]).
8
9   %closeLog.
10  closeLog :-
11    close(stream).
12  closeLog(A) :-
13    close(A).
14
15  %logger(_):-!.
16  logger(X):-
17    writeln(stream,X).
18
19  %logger1(_):-!.
20  logger1(X):-
21    write(stream,X).
22
23  logger_ice(X):-
24    writeln(stream,X).
25
26  % tronque le rÃ©sultat pour des positifs
27  truncate(X,N,Result):- X >= 0, Result is floor(10^N*X)/10^N, !.
28
29  %createLog(A) :-
30  % atomic_concat(A,".txt",FileName),
31  % atomic_concat("networks/systematic_test/logs/",FileName,CompleteName),
32  % open(CompleteName,append,MS,[alias(stream)]),
33  % writeln(MS,'Debut Log File').
34  createLog(A,StreamName) :-
35    atomic_concat(A,".txt",FileName),
36    atomic_concat("logs/",FileName,CompleteName),
37    open(CompleteName,append,MS,[alias(StreamName)]).
```

A.6 Tests

At the time of passing some tests, the final JSON is not available yet, and some options may be modified without notice. This does not impact the test results, as per non regression analysis.

A.6.1 Ground Truth

```
1  {"network":[
2  {"name":"test_tmp_5-2",
3  "borneMax":10,
4  "borneMin":0,
5  "borneEffectOnOthers":5,
6  "borneEffectOnSelf":0,
7  "globalThreshold":9,
8  "steps":8,
9  "method":"tmp",
10 "nodes":[
11 {"label":"gene1","type":"node"},
12 {"label":"gene2","type":"node"},
13 {"label":"gene3","type":"node"},
14 {"label":"gene4","type":"node"}],
15 "edges":[
16 {"id":"toChange","from":"gene1","to":"gene2","threshold":4,"borneEffect":5,"delay":0,"effect
       ":2},
17 {"id":"toChange","from":"gene2","to":"gene1","threshold":9,"borneEffect":5,"delay":0,"effect
       ":-1},
18 {"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect
       ":1},
19 {"id":"toChange","from":"gene3","to":"gene1","threshold":1,"borneEffect":5,"delay":0,"effect
       ":0},
20 {"id":"toChange","from":"gene2","to":"gene3","threshold":4,"borneEffect":5,"delay":0,"effect
       ":-1},
21 {"id":"toChange","from":"gene3","to":"gene2","threshold":1,"borneEffect":5,"delay":0,"effect
       ":0},
22 {"id":"toChange","from":"gene1","to":"gene4","threshold":4,"borneEffect":5,"delay":0,"effect
       ":-2},
```

```
{"id":"toChange","from":"gene4","to":"gene1","threshold":7,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"gene4","threshold":9,"borneEffect":5,"delay":0,"effect":2},
{"id":"toChange","from":"gene4","to":"gene2","threshold":1,"borneEffect":5,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":0}],
"data":[
{"node":"gene4","step":8,"niveau":0},
{"node":"gene4","step":7,"niveau":0},
{"node":"gene4","step":6,"niveau":0},
{"node":"gene4","step":5,"niveau":0},
{"node":"gene4","step":4,"niveau":1},
{"node":"gene4","step":3,"niveau":3},
{"node":"gene4","step":2,"niveau":5},
{"node":"gene4","step":1,"niveau":7},
{"node":"gene4","step":0,"niveau":9},
{"node":"gene3","step":8,"niveau":2},
{"node":"gene3","step":7,"niveau":2},
{"node":"gene3","step":6,"niveau":2},
{"node":"gene3","step":5,"niveau":2},
{"node":"gene3","step":4,"niveau":2},
{"node":"gene3","step":3,"niveau":2},
{"node":"gene3","step":2,"niveau":2},
{"node":"gene3","step":1,"niveau":1},
{"node":"gene3","step":0,"niveau":0},
{"node":"gene2","step":8,"niveau":10},
{"node":"gene2","step":7,"niveau":10},
{"node":"gene2","step":6,"niveau":10},
{"node":"gene2","step":5,"niveau":10},
{"node":"gene2","step":4,"niveau":8},
{"node":"gene2","step":3,"niveau":6},
{"node":"gene2","step":2,"niveau":4},
{"node":"gene2","step":1,"niveau":2},
{"node":"gene2","step":0,"niveau":0},
{"node":"gene1","step":8,"niveau":4},
{"node":"gene1","step":7,"niveau":5},
{"node":"gene1","step":6,"niveau":6},
{"node":"gene1","step":5,"niveau":7},
{"node":"gene1","step":4,"niveau":7},
{"node":"gene1","step":3,"niveau":7},
{"node":"gene1","step":2,"niveau":7},
{"node":"gene1","step":1,"niveau":6},
{"node":"gene1","step":0,"niveau":5}]}]}
```


A.6.2 Lac Operon


```
{"network":[
{"name":"Lac Operon",
"borneMax":10,
"borneMin":0,
"borneEffectOnOthers":3,
"borneEffectOnSelf":0,
"globalThreshold":2,
"steps":7,
"method":"tmp",
"sparsity":7,
"labeling":"all_ff",
"nSol":10,
"nodes":[
{"label":"operonAYZ","type":"node"},
{"label":"lactose","type":"node"},
{"label":"lacI","type":"node"},
{"label":"cap_cAMP","type":"node"},
{"label":"glucose","type":"node"},
{"label":"supportLacI","type":"control"}],
"edges":[
{"id":"toChange","from":"lactose","to":"lacI","threshold":"/","borneEffect":3,"delay":0,"effect":-2},
{"id":"toChange","from":"lacI","to":"operonAYZ","threshold":"/","borneEffect":3,"delay":0,"effect":-3},
{"id":"toChange","from":"operonAYZ","to":"glucose","threshold":"/","borneEffect":3,"delay":0,"effect":1},
```

```
{"id":"toChange","from":"operonAYZ","to":"lactose","threshold":"/","borneEffect":3,"delay":0,"effect":-2},
{"id":"toChange","from":"cap_cAMP","to":"operonAYZ","threshold":"/","borneEffect":3,"delay":0,"effect":2},
{"id":"toChange","from":"glucose","to":"cap_cAMP","threshold":"/","borneEffect":"/","delay":0,"effect":-2},
{"id":"toChange","from":"supportLacI","to":"lacI","threshold":"/","borneEffect":"/","delay":0,"effect":1},

{"id":"toChange","from":"glucose","to":"lacI","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"glucose","to":"lactose","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"glucose","to":"operonAYZ","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"operonAYZ","to":"lacI","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"operonAYZ","to":"cap_cAMP","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"lactose","to":"operonAYZ","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"lacI","to":"cap_cAMP","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"lacI","to":"lactose","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"lacI","to":"glucose","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"lactose","to":"cap_cAMP","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"lactose","to":"glucose","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"cap_cAMP","to":"lacI","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"cap_cAMP","to":"glucose","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"cap_cAMP","to":"lactose","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"supportLacI","to":"lactose","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"supportLacI","to":"glucose","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"supportLacI","to":"cap_cAMP","threshold":"/","borneEffect":1,"delay":0,"effect":0},
{"id":"toChange","from":"supportLacI","to":"operonAYZ","threshold":"/","borneEffect":1,"delay":0,"effect":0}
],
"data":[
{"node":"operonAYZ","step":0,"niveau":7},
{"node":"lacI","step":0,"niveau":2},
{"node":"cap_cAMP","step":0,"niveau":8},
{"node":"lactose","step":0,"niveau":9},
{"node":"glucose","step":0,"niveau":1},

{"node":"operonAYZ","step":3,"niveau":7},
{"node":"lacI","step":3,"niveau":0},
{"node":"cap_cAMP","step":3,"niveau":2},
{"node":"lactose","step":3,"niveau":3},
{"node":"glucose","step":3,"niveau":4},

{"node":"supportLacI","step":0,"niveau":1},
{"node":"supportLacI","step":1,"niveau":1},
{"node":"supportLacI","step":2,"niveau":1},
{"node":"supportLacI","step":3,"niveau":1},
{"node":"supportLacI","step":4,"niveau":1},
{"node":"supportLacI","step":5,"niveau":1},
{"node":"supportLacI","step":6,"niveau":1},
{"node":"supportLacI","step":7,"niveau":1}
]}]}
```


A.6.3 Sparsity Results


```
{"network":[{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json","borneMax":10,"borneMin":0,"borneEffectOnOthers":5,"borneEffectOnSelf":0,"globalThreshold":9,"steps":2,"method":"memoisation","nodes":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"},{"label":"gene3","type":"node"},{"label":"gene4","type":"node"}],"edges":[{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":5,"delay":0,"effect":2},{"id":"toChange","from":"gene2","to":"gene1","threshold":1,"borneEffect":0,"delay":0,"effect":0},
```

```
{"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene3","to":"gene1","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene2","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene2","threshold":2,"borneEffect":0,"delay":0,"effect":0},{"id":"
    toChange","from":"gene1","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":-2},{"id":"
    toChange","from":"gene4","to":"gene1","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"gene4","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene2","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene4","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0}],

"data":[{"node":"gene4","step":2,"niveau":5},{"node":"gene4","step":1,"niveau":7},{"node":"gene4","step":0,"
    niveau":9},{"node":"gene3","step":2,"niveau":2},{"node":"gene3","step":1,"niveau":1},{"node":"gene3","
    step":0,"niveau":0},{"node":"gene2","step":2,"niveau":4},{"node":"gene2","step":1,"niveau":2},{"node":"
    gene2","step":0,"niveau":0},{"node":"gene1","step":2,"niveau":7},{"node":"gene1","step":1,"niveau":6},{"
    node":"gene1","step":0,"niveau":5}]},{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json","
    borneMax":10,"borneMin":0,"borneEffectOnOthers":5,"borneEffectOnSelf":0,"globalThreshold":9,"steps":2,"
    method":"memoisation","nodes":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"},{"label":"
    gene3","type":"node"},{"label":"gene4","type":"node"}],"edges":[
{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":5,"delay":0,"effect":2},
{"id":"toChange","from":"gene2","to":"gene1","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene3","to":"gene1","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene2","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene2","threshold":2,"borneEffect":0,"delay":0,"effect":0},{"id":"
    toChange","from":"gene1","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":-2},{"id":"
    toChange","from":"gene4","to":"gene1","threshold":7,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"gene4","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene2","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene4","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0}],

"data":[{"node":"gene4","step":2,"niveau":5},{"node":"gene4","step":1,"niveau":7},{"node":"gene4","step":0,"
    niveau":9},{"node":"gene3","step":2,"niveau":2},{"node":"gene3","step":1,"niveau":1},{"node":"gene3","
    step":0,"niveau":0},{"node":"gene2","step":2,"niveau":4},{"node":"gene2","step":1,"niveau":2},{"node":"
    gene2","step":0,"niveau":0},{"node":"gene1","step":2,"niveau":7},{"node":"gene1","step":1,"niveau":6},{"
    node":"gene1","step":0,"niveau":5}]},{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json","
    borneMax":10,"borneMin":0,"borneEffectOnOthers":5,"borneEffectOnSelf":0,"globalThreshold":9,"steps":2,"
    method":"memoisation","nodes":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"},{"label":"
    gene3","type":"node"},{"label":"gene4","type":"node"}],"edges":[{"id":"toChange","from":"gene1","to":"
    gene2","threshold":1,"borneEffect":5,"delay":0,"effect":2},{"id":"toChange","from":"gene2","to":"gene1","
    threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene3","to":"gene1","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene2","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene2","threshold":2,"borneEffect":0,"delay":0,"effect":0},{"id":"
    toChange","from":"gene1","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":-2},{"id":"
    toChange","from":"gene4","to":"gene1","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"gene4","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene2","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene4","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0}],

"data":[{"node":"gene4","step":2,"niveau":5},{"node":"gene4","step":1,"niveau":7},{"node":"gene4","step":0,"
    niveau":9},{"node":"gene3","step":2,"niveau":2},{"node":"gene3","step":1,"niveau":1},{"node":"gene3","
    step":0,"niveau":0},{"node":"gene2","step":2,"niveau":4},{"node":"gene2","step":1,"niveau":2},{"node":"
    gene2","step":0,"niveau":0},{"node":"gene1","step":2,"niveau":7},{"node":"gene1","step":1,"niveau":6},{"
    node":"gene1","step":0,"niveau":5}]},{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json","
    borneMax":10,"borneMin":0,"borneEffectOnOthers":5,"borneEffectOnSelf":0,"globalThreshold":9,"steps":2,"
    method":"memoisation","nodes":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"},{"label":"
    gene3","type":"node"},{"label":"gene4","type":"node"}],
"edges":[
{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":5,"delay":0,"effect":2},
{"id":"toChange","from":"gene2","to":"gene1","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene3","to":"gene1","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene2","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene2","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":-2},{"id":"
    toChange","from":"gene4","to":"gene1","threshold":7,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"gene4","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene2","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene4","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0}],
"data":[{"node":"gene4","step":2,"niveau":5},{"node":"gene4","step":1,"niveau":7},{"node":"gene4","step":0,"
    niveau":9},{"node":"gene3","step":2,"niveau":2},{"node":"gene3","step":1,"niveau":1},{"node":"gene3","
    step":0,"niveau":0},{"node":"gene2","step":2,"niveau":4},{"node":"gene2","step":1,"niveau":2},{"node":"
    gene2","step":0,"niveau":0},{"node":"gene1","step":2,"niveau":7},{"node":"gene1","step":1,"niveau":6},{"
    node":"gene1","step":0,"niveau":5}]}
,{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json","borneMax":10,"borneMin":0,"
    borneEffectOnOthers":5,"borneEffectOnSelf":0,"globalThreshold":9,"steps":2,"method":"memoisation","nodes
    ":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"},{"label":"gene3","type":"node"},{"
    label":"gene4","type":"node"}],"edges":[
{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":5,"delay":0,"effect":2},
{"id":"toChange","from":"gene2","to":"gene1","threshold":3,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene3","to":"gene1","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene2","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene2","threshold":2,"borneEffect":0,"delay":0,"effect":0},{"id":"
    toChange","from":"gene1","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":-2},{"id":"
    toChange","from":"gene4","to":"gene1","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"gene4","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene2","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene4","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0}],
"data":[{"node":"gene4","step":2,"niveau":5},{"node":"gene4","step":1,"niveau":7},{"node":"gene4","step":0,"
    niveau":9},{"node":"gene3","step":2,"niveau":2},{"node":"gene3","step":1,"niveau":1},{"node":"gene3","
    step":0,"niveau":0},{"node":"gene2","step":2,"niveau":4},{"node":"gene2","step":1,"niveau":2},{"node":"
    gene2","step":0,"niveau":0},{"node":"gene1","step":2,"niveau":7},{"node":"gene1","step":1,"niveau":6},{"
    node":"gene1","step":0,"niveau":5}]},
{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json","borneMax":10,"borneMin":0,"
    borneEffectOnOthers":5,"borneEffectOnSelf":0,"globalThreshold":9,"steps":2,"method":"memoisation","nodes
    ":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"},{"label":"gene3","type":"node"},{"
```

```
label":"gene4","type":"node"}],"edges":
[
{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":5,"delay":0,"effect":2},
{"id":"toChange","from":"gene2","to":"gene1","threshold":3,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene3","to":"gene1","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene2","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene2","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":-2},
{"id":"toChange","from":"gene4","to":"gene1","threshold":7,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"gene4","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene2","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene4","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0}],
"data":[{"node":"gene4","step":2,"niveau":5},{"node":"gene4","step":1,"niveau":7},{"node":"gene4","step":0,"
    niveau":9},{"node":"gene3","step":2,"niveau":2},{"node":"gene3","step":1,"niveau":1},{"node":"gene3","
    step":0,"niveau":0},{"node":"gene2","step":2,"niveau":4},{"node":"gene2","step":1,"niveau":2},{"node":"
    gene2","step":0,"niveau":0},{"node":"gene1","step":2,"niveau":7},{"node":"gene1","step":1,"niveau":6},{"
    node":"gene1","step":0,"niveau":5}]},
{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json","borneMax":10,"borneMin":0,"
    borneEffectOnOthers":5,"borneEffectOnSelf":0,"globalThreshold":9,"steps":2,"method":"memoisation","nodes
    ":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"},{"label":"gene3","type":"node"},{"
    label":"gene4","type":"node"}],"edges":[
{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":5,"delay":0,"effect":2},
{"id":"toChange","from":"gene2","to":"gene1","threshold":3,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene3","to":"gene1","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene2","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene2","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":-2},
{"id":"toChange","from":"gene4","to":"gene1","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"gene4","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene2","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene4","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0}],
"data":[{"node":"gene4","step":2,"niveau":5},{"node":"gene4","step":1,"niveau":7},{"node":"gene4","step":0,"
    niveau":9},{"node":"gene3","step":2,"niveau":2},{"node":"gene3","step":1,"niveau":1},{"node":"gene3","
    step":0,"niveau":0},{"node":"gene2","step":2,"niveau":4},{"node":"gene2","step":1,"niveau":2},{"node":"
    gene2","step":0,"niveau":0},{"node":"gene1","step":2,"niveau":7},{"node":"gene1","step":1,"niveau":6},{"
    node":"gene1","step":0,"niveau":5}]},
{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json","borneMax":10,"borneMin":0,"
    borneEffectOnOthers":5,"borneEffectOnSelf":0,"globalThreshold":9,"steps":2,"method":"memoisation","nodes
    ":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"},{"label":"gene3","type":"node"},{"
    label":"gene4","type":"node"}],"edges":[
{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":5,"delay":0,"effect":2},
{"id":"toChange","from":"gene2","to":"gene1","threshold":3,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene3","to":"gene1","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene2","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene2","threshold":2,"borneEffect":0,"delay":0,"effect":0},{"id":"
    toChange","from":"gene1","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":-2},{"id":"
    toChange","from":"gene4","to":"gene1","threshold":7,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"gene4","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene2","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene4","threshold":2,"borneEffect":0,"delay":0,"effect":0},

{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0}],"data":[{"
    node":"gene4","step":2,"niveau":5},{"node":"gene4","step":1,"niveau":7},{"node":"gene4","step":0,"niveau
    ":9},{"node":"gene3","step":2,"niveau":2},{"node":"gene3","step":1,"niveau":1},{"node":"gene3","step":0,"
    niveau":0},{"node":"gene2","step":2,"niveau":4},{"node":"gene2","step":1,"niveau":2},{"node":"gene2","
    step":0,"niveau":0},{"node":"gene1","step":2,"niveau":7},{"node":"gene1","step":1,"niveau":6},{"node":"
    gene1","step":0,"niveau":5}]},{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json","
    borneMax":10,"borneMin":0,"borneEffectOnOthers":5,"borneEffectOnSelf":0,"globalThreshold":9,"steps":2,"
    method":"memoisation","nodes":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"},{"label":"
    gene3","type":"node"},{"label":"gene4","type":"node"}],"edges":[{"id":"toChange","from":"gene1","to":"
    gene2","threshold":1,"borneEffect":5,"delay":0,"effect":2},{"id":"toChange","from":"gene2","to":"gene1","
    threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":1},{"id":"
    toChange","from":"gene3","to":"gene1","threshold":2,"borneEffect":0,"delay":0,"effect":0},{"id":"toChange
    ","from":"gene2","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0},{"id":"toChange","from
    ":"gene3","to":"gene2","threshold":2,"borneEffect":0,"delay":0,"effect":0},{"id":"toChange","from":"gene1
    ","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":-2},{"id":"toChange","from":"gene4","to
    ":"gene1","threshold":1,"borneEffect":5,"delay":0,"effect":1},{"id":"toChange","from":"gene2","to":"gene4
    ","threshold":1,"borneEffect":0,"delay":0,"effect":0},{"id":"toChange","from":"gene4","to":"gene2","
    threshold":1,"borneEffect":0,"delay":0,"effect":0},{"id":"toChange","from":"gene3","to":"gene4","
    threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0}],
"data":[{"node":"gene4","step":2,"niveau":5},{"node":"gene4","step":1,"niveau":7},{"node":"gene4","step":0,"
    niveau":9},{"node":"gene3","step":2,"niveau":2},{"node":"gene3","step":1,"niveau":1},{"node":"gene3","
    step":0,"niveau":0},{"node":"gene2","step":2,"niveau":4},{"node":"gene2","step":1,"niveau":2},{"node":"
    gene2","step":0,"niveau":0},{"node":"gene1","step":2,"niveau":7},{"node":"gene1","step":1,"niveau":6},{"
    node":"gene1","step":0,"niveau":5}]},
{"name":"network_test_memoisation_steps2_nodes4_edges0_data0.json","borneMax":10,"borneMin":0,"
    borneEffectOnOthers":5,"borneEffectOnSelf":0,"globalThreshold":9,"steps":2,"method":"memoisation","nodes
    ":[{"label":"gene1","type":"node"},{"label":"gene2","type":"node"},{"label":"gene3","type":"node"},{"
    label":"gene4","type":"node"}],"edges":[
{"id":"toChange","from":"gene1","to":"gene2","threshold":1,"borneEffect":5,"delay":0,"effect":2},
{"id":"toChange","from":"gene2","to":"gene1","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene3","threshold":1,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene3","to":"gene1","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene2","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene2","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene1","to":"gene4","threshold":1,"borneEffect":5,"delay":0,"effect":-2},
{"id":"toChange","from":"gene4","to":"gene1","threshold":7,"borneEffect":5,"delay":0,"effect":1},
{"id":"toChange","from":"gene2","to":"gene4","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene2","threshold":1,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene3","to":"gene4","threshold":2,"borneEffect":0,"delay":0,"effect":0},
{"id":"toChange","from":"gene4","to":"gene3","threshold":1,"borneEffect":0,"delay":0,"effect":0}],
"data":[{"node":"gene4","step":2,"niveau":5},{"node":"gene4","step":1,"niveau":7},{"node":"gene4","step":0,"
    niveau":9},{"node":"gene3","step":2,"niveau":2},{"node":"gene3","step":1,"niveau":1},{"node":"gene3","
    step":0,"niveau":0},{"node":"gene2","step":2,"niveau":4},{"node":"gene2","step":1,"niveau":2},{"node":"
    gene2","step":0,"niveau":0},{"node":"gene1","step":2,"niveau":7},{"node":"gene1","step":1,"niveau":6},{"
    node":"gene1","step":0,"niveau":5}]}]}
```

A.6.4 Prolog

```
:- include(prolog_server).

% lancer la campagne de test : ?- systematic_test(all,"logfile_JJMMAAAA_HHMM").
systematic_test(A, General_logfile):-
  % Recupere la liste de tous les networks (fichiers)
  liste(A,Lclean),

  setup_call_cleanup(
    createLog(General_logfile,globalStream),
    systematic_test_rec(globalStream,Lclean),
    closeLog(globalStream)),!.

systematic_test_rec(GL,[]):-writeln("Done!").
systematic_test_rec(GL,[H|Tail]):-
    write(GL,H), write(GL, ","), get_time(T1), write(GL,T1),%
    (catch(call_with_time_limit(5,test(H)),Exception, true)->
          (var(Exception)-> !,
                  write("OK "),
                  get_time(T2),
                  T is T2-T1,
                  write(T),
                  writeln(" [s]"),
                  write(GL,",OK,"),
                  write(GL, T),
                  writeln(GL, ",[s]")
            ;     !,writeln("timeout"),writeln(GL, ",timeout,inf,[s]"))
    ;       !,
          get_time(T3),
          Tf is T3-T1,
          write(GL,",FAIL,"),
          write(GL, Tf),
          writeln(GL, ",[s]"),
          write(" ,FAIL, "),
          write(Tf),
          writeln("[s] ")),

    write("j'ai testÃ© => "), writeln(H),
    systematic_test_rec(GL,Tail).


test(Fichier_Test) :-
  setup_call_cleanup(
    createLog,
    test_(Fichier_Test),
    closeLog).

test_(Fichier_Test) :-logger("test"),
    atomic_concat("networks/systematic_test/",Fichier_Test,Fichier),
    open(Fichier, read, MonStream),
    json_read(MonStream,JsonIn),
    close(MonStream),
    logger(JsonIn),
    myTranslate(Name,
      BorneMaxNiveau,
      BorneMinNiveau,
      BorneEffectOnOthers,
      BorneEffectOnSelf,
      GlobalThreshold,
      Steps,
      Method,
      Sparsity,
      Labeling,
      NSol,
      Nodes,
      Edges,
      Data,
      JsonIn),
    logger("[test] Before Solve"),
        solve(Method,Sparsity,Labeling,Nodes, Edges,Data,BorneMinNiveau,BorneMaxNiveau,
            BorneEffectOnOthers,BorneEffectOnSelf,GlobalThreshold,Steps,Lassoc),
        logger("[test] After Solve"),
    get_data_edge(Lassoc,Data_output,Edges_output),
        produceJson(
          Name,
```

```
74          BorneMaxNiveau,
75          BorneMinNiveau,
76          BorneEffectOnOthers,
77          BorneEffectOnSelf,
78          GlobalThreshold,
79          Steps,
80          Method,
81          Sparsity,
82          Labeling,
83          NSol,
84          Nodes,
85          Edges_output,
86          Data_output,
87          DictOut),
88        logger("[test] Done").
89
90
91  nettoyer_liste(L,Lclean):-
92    nettoyer_liste_(L,Lclean,[]).
93
94  nettoyer_liste_([],Acc,Acc):-!.
95  nettoyer_liste_([H|T], Lclean,Acc):-
96    file_name_extension(_, json, H),
97    nettoyer_liste_(T,Lclean,[H|Acc]).
98  nettoyer_liste_([H|T],Lclean,Acc):-
99    nettoyer_liste_(T,Lclean,Acc).
100
101 liste(all,L):-
102   directory_files("networks/systematic_test/",Ltmp),
103   nettoyer_liste(Ltmp,L).
104
105 liste(lineaire,Lcleanlin):-
106   Lcleanlin =["network_test_lineaire_steps2_nodes2_edges0_data1.json", "
          network_test_lineaire_steps2_nodes2_edges0_data2.json", "
          network_test_lineaire_steps2_nodes2_edges0_data3.json", "
          network_test_lineaire_steps2_nodes2_edges2_data2.json", "
          network_test_lineaire_steps2_nodes2_edges2_data3.json", "
          network_test_lineaire_steps2_nodes3_edges0_data1.json",  "
          network_test_lineaire_steps2_nodes3_edges0_data2.json", "
          network_test_lineaire_steps2_nodes3_edges0_data3.json", "
          network_test_lineaire_steps2_nodes3_edges2_data1.json", "
          network_test_lineaire_steps2_nodes3_edges2_data2.json", "
          network_test_lineaire_steps2_nodes3_edges2_data3.json", "
          network_test_lineaire_steps2_nodes4_edges0_data0.json", "
          network_test_lineaire_steps2_nodes4_edges0_data1.json", "
          network_test_lineaire_steps2_nodes4_edges0_data2.json", "
          network_test_lineaire_steps2_nodes4_edges0_data3.json", "
          network_test_lineaire_steps2_nodes4_edges2_data0.json", "
          network_test_lineaire_steps2_nodes4_edges2_data1.json", "
          network_test_lineaire_steps2_nodes4_edges2_data2.json", "
          network_test_lineaire_steps2_nodes4_edges2_data3.json", "
          network_test_lineaire_steps2_nodes4_edges4_data0.json", "
          network_test_lineaire_steps2_nodes4_edges4_data1.json", "
          network_test_lineaire_steps2_nodes4_edges4_data2.json", "
          network_test_lineaire_steps2_nodes4_edges4_data3.json", "
          network_test_lineaire_steps2_nodes4_edges6_data0.json", "
          network_test_lineaire_steps2_nodes4_edges6_data1.json", "
          network_test_lineaire_steps2_nodes4_edges6_data2.json", "
          network_test_lineaire_steps2_nodes4_edges6_data3.json", "
          network_test_lineaire_steps4_nodes2_edges0_data2.json", "
          network_test_lineaire_steps4_nodes2_edges0_data3.json", "
          network_test_lineaire_steps4_nodes3_edges0_data1.json", "
          network_test_lineaire_steps4_nodes3_edges2_data2.json", "
          network_test_lineaire_steps4_nodes4_edges0_data0.json", "
          network_test_lineaire_steps4_nodes4_edges0_data2.json", "
          network_test_lineaire_steps4_nodes4_edges2_data0.json", "
          network_test_lineaire_steps4_nodes4_edges2_data1.json", "
          network_test_lineaire_steps4_nodes4_edges4_data0.json", "
          network_test_lineaire_steps4_nodes4_edges6_data0.json", "
          network_test_lineaire_steps6_nodes2_edges0_data3.json", "
          network_test_lineaire_steps6_nodes4_edges0_data0.json", "
          network_test_lineaire_steps6_nodes4_edges2_data0.json", "
          network_test_lineaire_steps6_nodes4_edges2_data1.json", "
          network_test_lineaire_steps6_nodes4_edges4_data0.json", "
          network_test_lineaire_steps6_nodes4_edges6_data0.json", "
          network_test_lineaire_steps8_nodes4_edges0_data0.json", "
```

```
        network_test_lineaire_steps8_nodes4_edges2_data0.json", "
        network_test_lineaire_steps8_nodes4_edges4_data0.json", "
        network_test_lineaire_steps8_nodes4_edges6_data0.json"].
107
108 liste(tmp,Lcleantmp):-
109   Lcleantmp = ["network_test_tmp_steps2_nodes2_edges0_data1.json", "
        network_test_tmp_steps2_nodes2_edges0_data2.json", "
        network_test_tmp_steps2_nodes2_edges0_data3.json", "
        network_test_tmp_steps2_nodes2_edges2_data2.json", "
        network_test_tmp_steps2_nodes2_edges2_data3.json", "
        network_test_tmp_steps2_nodes3_edges0_data1.json", "
        network_test_tmp_steps2_nodes3_edges0_data2.json", "
        network_test_tmp_steps2_nodes3_edges0_data3.json", "
        network_test_tmp_steps2_nodes3_edges2_data1.json", "
        network_test_tmp_steps2_nodes3_edges2_data2.json", "
        network_test_tmp_steps2_nodes3_edges2_data3.json", "
        network_test_tmp_steps2_nodes4_edges0_data0.json", "
        network_test_tmp_steps2_nodes4_edges0_data1.json", "
        network_test_tmp_steps2_nodes4_edges0_data2.json", "
        network_test_tmp_steps2_nodes4_edges0_data3.json", "
        network_test_tmp_steps2_nodes4_edges2_data0.json", "
        network_test_tmp_steps2_nodes4_edges2_data1.json", "
        network_test_tmp_steps2_nodes4_edges2_data2.json", "
        network_test_tmp_steps2_nodes4_edges2_data3.json", "
        network_test_tmp_steps2_nodes4_edges4_data0.json", "
        network_test_tmp_steps2_nodes4_edges4_data1.json", "
        network_test_tmp_steps2_nodes4_edges4_data2.json", "
        network_test_tmp_steps2_nodes4_edges4_data3.json", "
        network_test_tmp_steps2_nodes4_edges6_data0.json", "
        network_test_tmp_steps2_nodes4_edges6_data1.json", "
        network_test_tmp_steps2_nodes4_edges6_data2.json", "
        network_test_tmp_steps2_nodes4_edges6_data3.json", "
        network_test_tmp_steps4_nodes2_edges0_data2.json", "
        network_test_tmp_steps4_nodes2_edges0_data3.json", "
        network_test_tmp_steps4_nodes3_edges0_data2.json", "
        network_test_tmp_steps4_nodes3_edges2_data2.json", "
        network_test_tmp_steps4_nodes4_edges0_data1.json", "
        network_test_tmp_steps4_nodes4_edges2_data1.json", "
        network_test_tmp_steps4_nodes4_edges2_data2.json", "
        network_test_tmp_steps4_nodes4_edges2_data3.json", "
        network_test_tmp_steps4_nodes4_edges4_data0.json", "
        network_test_tmp_steps4_nodes4_edges4_data1.json", "
        network_test_tmp_steps4_nodes4_edges4_data3.json", "
        network_test_tmp_steps4_nodes4_edges6_data0.json", "
        network_test_tmp_steps4_nodes4_edges6_data1.json", "
        network_test_tmp_steps4_nodes4_edges6_data2.json", "
        network_test_tmp_steps4_nodes4_edges6_data3.json", "
        network_test_tmp_steps6_nodes2_edges0_data3.json", "
        network_test_tmp_steps6_nodes4_edges4_data0.json", "
        network_test_tmp_steps6_nodes4_edges6_data0.json", "
        network_test_tmp_steps8_nodes4_edges4_data0.json", "
        network_test_tmp_steps8_nodes4_edges6_data0.json"].
110
111 liste(memoisation,Lcleanmem):-
112   Lcleanmem = ["network_test_memoisation_steps2_nodes2_edges0_data1.json", "
        network_test_memoisation_steps2_nodes2_edges0_data2.json", "
        network_test_memoisation_steps2_nodes2_edges0_data3.json", "
        network_test_memoisation_steps2_nodes2_edges2_data2.json", "
        network_test_memoisation_steps2_nodes2_edges2_data3.json", "
        network_test_memoisation_steps2_nodes3_edges0_data1.json", "
        network_test_memoisation_steps2_nodes3_edges0_data2.json", "
        network_test_memoisation_steps2_nodes3_edges0_data3.json", "
        network_test_memoisation_steps2_nodes3_edges2_data1.json", "
        network_test_memoisation_steps2_nodes3_edges2_data2.json", "
        network_test_memoisation_steps2_nodes3_edges2_data3.json", "
        network_test_memoisation_steps2_nodes4_edges0_data0.json", "
        network_test_memoisation_steps2_nodes4_edges0_data1.json", "
        network_test_memoisation_steps2_nodes4_edges0_data2.json", "
        network_test_memoisation_steps2_nodes4_edges0_data3.json", "
        network_test_memoisation_steps2_nodes4_edges2_data0.json", "
        network_test_memoisation_steps2_nodes4_edges2_data1.json", "
        network_test_memoisation_steps2_nodes4_edges2_data2.json", "
        network_test_memoisation_steps2_nodes4_edges2_data3.json", "
        network_test_memoisation_steps2_nodes4_edges4_data0.json", "
        network_test_memoisation_steps2_nodes4_edges4_data1.json", "
        network_test_memoisation_steps2_nodes4_edges4_data2.json", "
```

```
        network_test_memoisation_steps2_nodes4_edges4_data3.json", "
        network_test_memoisation_steps2_nodes4_edges6_data0.json", "
        network_test_memoisation_steps2_nodes4_edges6_data1.json", "
        network_test_memoisation_steps2_nodes4_edges6_data2.json", "
        network_test_memoisation_steps2_nodes4_edges6_data3.json", "
        network_test_memoisation_steps4_nodes2_edges0_data2.json", "
        network_test_memoisation_steps4_nodes2_edges0_data3.json", "
        network_test_memoisation_steps4_nodes3_edges0_data2.json", "
        network_test_memoisation_steps4_nodes3_edges2_data2.json", "
        network_test_memoisation_steps4_nodes4_edges0_data1.json", "
        network_test_memoisation_steps4_nodes4_edges2_data1.json", "
        network_test_memoisation_steps4_nodes4_edges2_data2.json", "
        network_test_memoisation_steps4_nodes4_edges2_data3.json", "
        network_test_memoisation_steps4_nodes4_edges4_data0.json", "
        network_test_memoisation_steps4_nodes4_edges4_data1.json", "
        network_test_memoisation_steps4_nodes4_edges4_data2.json", "
        network_test_memoisation_steps4_nodes4_edges4_data3.json", "
        network_test_memoisation_steps4_nodes4_edges6_data0.json", "
        network_test_memoisation_steps4_nodes4_edges6_data1.json", "
        network_test_memoisation_steps4_nodes4_edges6_data2.json", "
        network_test_memoisation_steps4_nodes4_edges6_data3.json", "
        network_test_memoisation_steps6_nodes2_edges0_data3.json", "
        network_test_memoisation_steps6_nodes4_edges4_data0.json", "
        network_test_memoisation_steps6_nodes4_edges6_data0.json", "
        network_test_memoisation_steps8_nodes4_edges4_data0.json", "
        network_test_memoisation_steps8_nodes4_edges6_data0.json" ].
```

A.6.5 Python scripts

```
1  #!/usr/bin/python
2  import os
3
4  #-------------------------------------
5  #to change according to the test done
6  #-------------------------------------
7  source = open("logs/test_logfile2.txt","r").
8
9  text_source = source.readlines()
10 text_source.sort()
11
12
13 # Creation des fichiers cibles
14 targetMemoisation = open("results/memoisation.txt","w")
15 targetLineaire = open("results/lineaire.txt","w")
16 targetTmp = open("results/tmp.txt","w")
17 targetAsync = open("results/async.txt","w")
18 targetTable = open("results/table.txt","w")
19 summary = open("results/summary.txt","w")
20 # Initialisation des fichiers cibles
21 for target in [targetMemoisation, targetLineaire, targetTmp, targetAsync]:
22   target.write("[")
23
24
25 counter_fail_tmp = 0
26 counter_fail_lin = 0
27 counter_fail_memo = 0
28 counter_fail_async = 0
29
30 counter_timeout_tmp = 0
31 counter_timeout_lin = 0
32 counter_timeout_memo = 0
33 counter_timeout_async = 0
34 ind_i=0
35
36 nomTestList=[]
37 linRes=[]
38 tmpRes=[]
39 memRes=[]
40 asyRes=[]
41 linTime=[]
42 tmpTime=[]
43 memTime=[]
44 asyTime=[]
45
46 targetTable.write("|␣METHOD␣|␣STEPS␣|␣NODES␣|␣EDGES␣|␣DATA␣|␣RESULT␣|␣\n")
```

```
47 summary.write("|"+"TEST".center(12) +"|"+"LIN␣([ms])".center(12) + "|"+ "TMP␣([ms])".center
       (12) + "|" + "MEM␣([ms])".center(12) + "|" + "\n")# ASY (     ) \n")
48
49 #network_test_asynchronous_steps8_nodes3_edges0_data0.json
50
51 for line in text_source:
52   if(line[0] != "%" and len(line) > 2 ):
53     splitLine = line.split(",")
54     nomTest = splitLine[0]
55     #print(nomTest)
56     nomSplit = nomTest.split("_")
57     #print(nomSplit)
58     method = nomSplit[2]
59     steps = nomSplit[3][5:]
60     nodes = nomSplit[4][5:]
61     edges = nomSplit[5][5:]
62     data = nomSplit[6][4:].split(".")[0]
63
64     logfile = splitLine[1]
65     result =splitLine[2]
66     timing =splitLine[3]
67     time_ms = round(float(timing), 3)*1000
68
69     if result=="timeout":
70       res="/"
71     if result=="FAIL":
72       res="0"
73     if result=="OK":
74       res="1"
75
76     if method[0:3]=="lin":
77       linRes.append(res)
78       linTime.append(str(time_ms))
79     elif method[0:3]=="tmp":
80       tmpRes.append(res)
81       tmpTime.append(str(time_ms))
82     elif method[0:3]=="mem":
83       memRes.append(res)
84       memTime.append(str(time_ms))
85     elif method[0:3]=="asy":
86       asyRes.append(res)
87       asyTime.append(str(time_ms))
88
89     strName=str(steps)+"-"+str(nodes)+"-"+str(edges)+"-"+str(data)
90     if(not(strName in nomTestList)):
91       nomTestList.append(strName)
92
93     #if result==" timeout":
94     # time = splitLine[3]
95     if (result=="timeout" and method=="asynchronous"):
96       counter_timeout_async+=1
97     if (result=="timeout" and method=="tmp"):
98       counter_timeout_tmp+=1
99     if (result=="timeout" and method=="lineaire"):
100      counter_timeout_lin+=1
101    if (result=="timeout" and method=="memoisation"):
102      counter_timeout_memo+=1
103
104    if (result=="FAIL" and method=="asynchronous"):
105      counter_fail_async+=1
106      targetAsync.write("\"" + nomTest+"\",")
107    if (result=="FAIL" and method=="tmp"):
108      counter_fail_tmp+=1
109      targetTmp.write("\"" + nomTest+"\",")
110    if (result=="FAIL" and method=="lineaire"):
111      counter_fail_lin+=1
112      targetLineaire.write("\"" + nomTest+"\",")
113    if (result=="FAIL" and method=="memoisation"):
114      counter_fail_memo+=1
115      targetMemoisation.write("\"" + nomTest+"\",")
116
117    targetTable.write("|"+method[0:3].center(12)+"|"+steps.center(12)+"|"+nodes.center(12)+"|"
         +edges.center(12)+"|"+data.center(12)+"|"+res.center(12)+"|\n")
118
119 for i in range(0,len(nomTestList)-1):
120   #print(i)
```

```
121     summary.write("|"+nomTestList[i].center(12)+"|"+ (linRes[i]+"("+linTime[i] +")").center(12)
            + "|"+ (tmpRes[i]+"("+tmpTime[i] +")").center(12)+ "|"+ (memRes[i]+"("+memTime[i] +")").
            center(12) + "|\n")#asyRes[i]+"("+ +")   / \n")
122
123   print("FAILED  LIN - TMP - MEMO - ASYNC ==> " + str(counter_fail_lin) + "  --  " + str(
          counter_fail_tmp) + "  --  " + str(counter_fail_memo) + "  --  " + str(counter_fail_async))
124   print("TIMEOUT LIN - TMP - MEMO - ASYNC ==> " + str(counter_timeout_lin) + "  --  " + str(
          counter_timeout_tmp) + "  --  " + str(counter_timeout_memo) + "  --  " + str(
          counter_timeout_async))
125
126   # Fermetures des fichiers cibles
127   source.close()
128   targetTable.close()
129   summary.close()
130   for target in [targetMemoisation, targetLineaire, targetTmp, targetAsync]:
131     target.write("]")
132     target.close()


  1   #!/usr/bin/python
  2   import os
  3   from numpy import *
  4   #-------------------------------------
  5   #to change according to the test done
  6   #-------------------------------------
  7   method="tmp"
  8
  9   source_labeling_reverse = open("logs/essai_labeling_reverse_"+ method +".txt","r")
 10   source_labeling_all_ff = open("logs/essai_labeling_all_ff_"+ method +".txt","r")
 11   source_labeling_all_std = open("logs/essai_labeling_all_std_"+ method +".txt","r")
 12   source_labeling_norm = open("logs/essai_labeling_norm_"+ method +".txt","r")
 13
 14   text_source_labeling_norm = source_labeling_norm.readlines()
 15   text_source_labeling_all_std = source_labeling_all_std.readlines()
 16   text_source_labeling_all_ff = source_labeling_all_ff.readlines()
 17   text_source_labeling_reverse = source_labeling_reverse.readlines()
 18
 19   text_source_labeling_norm.sort()
 20   text_source_labeling_all_std.sort()
 21   text_source_labeling_all_ff.sort()
 22   text_source_labeling_reverse.sort()
 23
 24   # Creation des fichiers cibles
 25   summary_labeling = open("results/summary_labeling_"+method+".txt","w")
 26   # Initialisation des fichiers cibles
 27   for target in [summary_labeling]:
 28     target.write("
          %-------------------------------------------------------------------------------------------------------------------------------%\
          n")
 29     target.write("%-----------------------------------AUTOMATICALLY GENERATED RESULT FILE
          -----------------------------%\n")
 30     target.write("%-------------------------------------------USING labeling_testbench.py
          -----------------------------%\n")
 31     target.write("
          %-------------------------------------------------------------------------------------------------------------------------------%\
          n")
 32
 33
 34   results=[[],[],[],[],[]]
 35   summary_labeling.write( "|"+ "TEST".center(43) + "|" + "ALL-std[ms]".center(11) +"|"+ "ALL-ff
          [ms]".center(11) +"|"+ "NORM [ms]".center(11) + "|"+"REV [ms]".center(11) +"|\n")
 36
 37   # network_test_lineaire_steps2_nodes2_edges0_data1.json,1534178956.3300853,OK
          ,0.0086669921875,[s]
 38   # network_test_lineaire_steps8_nodes3_edges0_data0.json
 39
 40   # ce sont les memes noms par methode appliquees
 41   for line in text_source_labeling_norm:
 42     if(line[0] != "%" and len(line) > 2 ):
 43       splitLine = line.split(",")
 44       nomTest = splitLine[0]
 45       nomSplit = nomTest.split("_")
 46       results[0].append(nomTest[13:])
 47
 48
 49   indice_colonne=1
 50   for text_source in [text_source_labeling_all_std, text_source_labeling_all_ff,
```

```
    text_source_labeling_norm, text_source_labeling_reverse]:
  for line in text_source:
    if(line[0] != "%" and len(line) > 2 ):
      splitLine = line.split(",")
      timing =splitLine[3]
      time_ms = round(float(timing), 3)*1000
      time_str = str(time_ms)
      results[indice_colonne].append(time_str)
  indice_colonne+=1



for i in range(0,len(results[0])-1):
  #print(i)
  summary_labeling.write("|"+results[0][i].center(43)+"|"+results[1][i].center(11)+"|"+results
      [2][i].center(11)+"|"+results[3][i].center(11)+"|" + results[4][i].center(11)+"|" +"\n")


# Fermetures des fichiers cibles
source_labeling_reverse.close()
source_labeling_all_std.close()
source_labeling_all_ff.close()
source_labeling_norm.close()
summary_labeling.close()
```

```
#!/usr/bin/python
import os

# suppression des fichiers
dirdest = os.getcwd()
nodeList = ["gene1","gene2","gene3","gene4","gene5","gene5","gene7","gene8","gene9","gene10"]
edgeList = [
"{\"id\":\"tochange\",\"from\":\"gene1\",\"to\":\"gene2\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":2}",
"{\"id\":\"tochange\",\"from\":\"gene2\",\"to\":\"gene1\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":-1}",
"{\"id\":\"tochange\",\"from\":\"gene1\",\"to\":\"gene3\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":1}",
"{\"id\":\"tochange\",\"from\":\"gene2\",\"to\":\"gene4\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":2}",
"{\"id\":\"tochange\",\"from\":\"gene4\",\"to\":\"gene1\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":1}",
"{\"id\":\"tochange\",\"from\":\"gene3\",\"to\":\"gene1\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":0}",
"{\"id\":\"tochange\",\"from\":\"gene3\",\"to\":\"gene4\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":0}",
"{\"id\":\"tochange\",\"from\":\"gene3\",\"to\":\"gene2\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":0}"]

edgeList_full = [
"{\"id\":\"tochange\",\"from\":\"gene1\",\"to\":\"gene2\",\"threshold\":4,\"borneEffect\":5,\"
    delay\":0,\"effect\":2}",
"{\"id\":\"tochange\",\"from\":\"gene2\",\"to\":\"gene1\",\"threshold\":9,\"borneEffect\":5,\"
    delay\":0,\"effect\":-1}",
"{\"id\":\"tochange\",\"from\":\"gene1\",\"to\":\"gene3\",\"threshold\":1,\"borneEffect\":5,\"
    delay\":0,\"effect\":1}",
"{\"id\":\"tochange\",\"from\":\"gene2\",\"to\":\"gene4\",\"threshold\":9,\"borneEffect\":5,\"
    delay\":0,\"effect\":2}",
"{\"id\":\"tochange\",\"from\":\"gene4\",\"to\":\"gene1\",\"threshold\":7,\"borneEffect\":5,\"
    delay\":0,\"effect\":1}",
"{\"id\":\"tochange\",\"from\":\"gene3\",\"to\":\"gene1\",\"threshold\":1,\"borneEffect\":5,\"
    delay\":0,\"effect\":0}",
"{\"id\":\"tochange\",\"from\":\"gene3\",\"to\":\"gene4\",\"threshold\":1,\"borneEffect\":5,\"
    delay\":0,\"effect\":0}",
"{\"id\":\"tochange\",\"from\":\"gene3\",\"to\":\"gene2\",\"threshold\":1,\"borneEffect\":5,\"
    delay\":0,\"effect\":0}"]


dataList = [
["{\"node\":\"gene4\",\"step\":8,\"niveau\":0}","{\"node\":\"gene3\",\"step\":8,\"niveau\":2}"
    ,"{\"node\":\"gene2\",\"step\":8,\"niveau\":10}","{\"node\":\"gene1\",\"step\":8,\"niveau
    \":4}"],
["{\"node\":\"gene4\",\"step\":7,\"niveau\":0}","{\"node\":\"gene3\",\"step\":7,\"niveau\":2}"
    ,"{\"node\":\"gene2\",\"step\":7,\"niveau\":10}","{\"node\":\"gene1\",\"step\":7,\"niveau
    \":5}"],["{\"node\":\"gene4\",\"step\":6,\"niveau\":0}","{\"node\":\"gene3\",\"step\":6,\"
    niveau\":2}","{\"node\":\"gene2\",\"step\":6,\"niveau\":10}","{\"node\":\"gene1\",\"step
```

```
    \":6,\"niveau\":6}"],["{\"node\":\"gene4\",\"step\":5,\"niveau\":0}","{\"node\":\"gene3
    \",\"step\":5,\"niveau\":2}","{\"node\":\"gene2\",\"step\":5,\"niveau\":10}","{\"node\":\"
    gene1\",\"step\":5,\"niveau\":7}"],["{\"node\":\"gene4\",\"step\":4,\"niveau\":1}","{\"
    node\":\"gene3\",\"step\":4,\"niveau\":2}","{\"node\":\"gene2\",\"step\":4,\"niveau\":8}",
    "{\"node\":\"gene1\",\"step\":4,\"niveau\":7}"],["{\"node\":\"gene4\",\"step\":3,\"niveau
    \":3}","{\"node\":\"gene3\",\"step\":3,\"niveau\":2}","{\"node\":\"gene2\",\"step\":3,\"
    niveau\":6}","{\"node\":\"gene1\",\"step\":3,\"niveau\":7}"],["{\"node\":\"gene4\",\"step
    \":2,\"niveau\":5}","{\"node\":\"gene3\",\"step\":2,\"niveau\":2}","{\"node\":\"gene2\",\"
    step\":2,\"niveau\":4}","{\"node\":\"gene1\",\"step\":2,\"niveau\":7}"],["{\"node\":\"
    gene4\",\"step\":1,\"niveau\":7}","{\"node\":\"gene3\",\"step\":1,\"niveau\":1}","{\"node
    \":\"gene2\",\"step\":1,\"niveau\":2}","{\"node\":\"gene1\",\"step\":1,\"niveau\":6}"],["
    {\"node\":\"gene4\",\"step\":0,\"niveau\":9}","{\"node\":\"gene3\",\"step\":0,\"niveau
    \":0}","{\"node\":\"gene2\",\"step\":0,\"niveau\":0}","{\"node\":\"gene1\",\"step\":0,\"
    niveau\":5}"]]

edgeList_bool = [
"{\"id\":\"tochange\",\"from\":\"gene1\",\"to\":\"gene2\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":1}",
"{\"id\":\"tochange\",\"from\":\"gene2\",\"to\":\"gene1\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":-1}",
"{\"id\":\"tochange\",\"from\":\"gene1\",\"to\":\"gene3\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":1}",
"{\"id\":\"tochange\",\"from\":\"gene2\",\"to\":\"gene4\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":1}",
"{\"id\":\"tochange\",\"from\":\"gene4\",\"to\":\"gene1\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":1}",
"{\"id\":\"tochange\",\"from\":\"gene3\",\"to\":\"gene1\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":0}",
"{\"id\":\"tochange\",\"from\":\"gene3\",\"to\":\"gene4\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":0}",
"{\"id\":\"tochange\",\"from\":\"gene3\",\"to\":\"gene2\",\"threshold\":\"/\",\"borneEffect
    \":\"/\",\"delay\":0,\"effect\":0}"]

dataList_bool = [
["{\"node\":\"gene4\",\"step\":8,\"niveau\":0}","{\"node\":\"gene3\",\"step\":8,\"niveau\":2}"
    ,"{\"node\":\"gene2\",\"step\":8,\"niveau\":2}","{\"node\":\"gene1\",\"step\":8,\"niveau
    \":0}"],
["{\"node\":\"gene4\",\"step\":7,\"niveau\":0}","{\"node\":\"gene3\",\"step\":7,\"niveau\":2}"
    ,"{\"node\":\"gene2\",\"step\":7,\"niveau\":2}","{\"node\":\"gene1\",\"step\":7,\"niveau
    \":1}"],["{\"node\":\"gene4\",\"step\":6,\"niveau\":0}","{\"node\":\"gene3\",\"step\":6,\"
    niveau\":2}","{\"node\":\"gene2\",\"step\":6,\"niveau\":2}","{\"node\":\"gene1\",\"step
    \":6,\"niveau\":1}"],["{\"node\":\"gene4\",\"step\":5,\"niveau\":0}","{\"node\":\"gene3
    \",\"step\":5,\"niveau\":2}","{\"node\":\"gene2\",\"step\":5,\"niveau\":2}","{\"node\":\"
    gene1\",\"step\":5,\"niveau\":1}"],["{\"node\":\"gene4\",\"step\":4,\"niveau\":1}","{\"
    node\":\"gene3\",\"step\":4,\"niveau\":2}","{\"node\":\"gene2\",\"step\":4,\"niveau\":2}",
    "{\"node\":\"gene1\",\"step\":4,\"niveau\":1}"],["{\"node\":\"gene4\",\"step\":3,\"niveau
    \":2}","{\"node\":\"gene3\",\"step\":3,\"niveau\":2}","{\"node\":\"gene2\",\"step\":3,\"
    niveau\":2}","{\"node\":\"gene1\",\"step\":3,\"niveau\":2}"],["{\"node\":\"gene4\",\"step
    \":2,\"niveau\":2}","{\"node\":\"gene3\",\"step\":2,\"niveau\":2}","{\"node\":\"gene2\",\"
    step\":2,\"niveau\":2}","{\"node\":\"gene1\",\"step\":2,\"niveau\":2}"],["{\"node\":\"
    gene4\",\"step\":1,\"niveau\":2}","{\"node\":\"gene3\",\"step\":1,\"niveau\":1}","{\"node
    \":\"gene2\",\"step\":1,\"niveau\":1}","{\"node\":\"gene1\",\"step\":1,\"niveau\":2}"],["
    {\"node\":\"gene4\",\"step\":0,\"niveau\":2}","{\"node\":\"gene3\",\"step\":0,\"niveau
    \":0}","{\"node\":\"gene2\",\"step\":0,\"niveau\":0}","{\"node\":\"gene1\",\"step\":0,\"
    niveau\":2}"]]

for fichier in os.listdir(dirdest):
    if 'network_test' in fichier:
        os.remove(os.path.join(dirdest,fichier))

for method in ["lineaire", "tmp", "memoisation"]:#, "asynchronous"]:
  for steps in [2,4,6,8]:
    for nbreNode in [2,3,4]:
      for nbreEdge in range(0, 2*nbreNode, 2):
        # nbreData represente le nombre de donnees skippee entre deux data. 0 signifie que
            toutes les data sont donnees.
        for nbreData in [0,1,2,3]:

          filename = "network_test_"+method+"_steps"+str(steps)+"_nodes"+str(nbreNode)+"_edges
              "+str(nbreEdge)+"_data"+str(nbreData)+".json"
          node_i = 0
          nodeStr = ""
          while node_i <nbreNode:
            gene = nodeList[node_i]
            nodeStr += "{\"label\":\""+gene+"\",␣\"type\":\"node\"}"
            if node_i != (nbreNode-1):
```

```
64              nodeStr +=","
65            node_i+=1
66
67          edge_i = 0
68          edgeStr =""
69
70          while edge_i <nbreEdge:
71            edge_to_add = edgeList[edge_i]
72            edgeStr += edge_to_add
73            if edge_i != (nbreEdge-1):
74              edgeStr +=","
75            edge_i+=1
76
77          node_j = 0
78          step_i = 0
79          dataStr = ""
80          # Permet de produire un string contenant les donnees completes adaptees a la duree.
81          # node_j : from 0 -> 3
82          while node_j <nbreNode:
83            step_i=0
84          # step_i : from 0 -> 8
85            while step_i <= steps:
86              dataStr += dataList[8-step_i][3-node_j]
87              dataStr += ","
88              step_i +=(1+nbreData)
89            node_j +=1
90          dataStr = dataStr[:-1]
91          print(nbreNode)
92          print(nbreEdge)
93          print(dataStr)
94
95          target = open(filename,"w")
96          borneMax=10
97          borneMin=0
98          borneEffectOnOthers = 5
99          borneEffectOnSelf=0
100         globalThreshold=9
101         mySteps=steps
102         sparsity=0
103         labeling="all_ff"
104         nSol=1
105         content="{\"network\":␣␣[{\"name\":\"" + filename +"\",\"borneMax\":"+str(borneMax)+
                ",␣␣\"borneMin\":"+str(borneMin)+",\"borneEffectOnOthers\":␣"+ str(
                borneEffectOnOthers)+",\"borneEffectOnSelf\":"+str(borneEffectOnSelf) + ",\"
                globalThreshold\":"+str(globalThreshold)+",\"steps\":"+str(mySteps)+",\"method
                \":\""+method +"\",␣\"sparsity\":"+str(sparsity)+",\"labeling\":\""+labeling +"
                \",\"nSol\":"+str(nSol)+",\"nodes\":["+nodeStr+"],\"edges\":␣["+edgeStr+"],\"
                data\":␣["+ dataStr + "]}]}"
106         target.write(content)
107         target.close()
```

A.7 Front-End

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <link rel="stylesheet" href="css/main.css">
6      <title>GeneTool</title>
7      <script>
8          var Matrices=[];
9          // 1. create a new XMLHttpRequest object -- an object like any other!
10         var myRequest = new XMLHttpRequest();
11         // 2. open the request and pass the HTTP method name and the resource as
               parameters
12         // 3. write a function that runs anytime the state of the AJAX request changes
13         myRequest.onreadystatechange = function () {
14             // 4. check if the request has a readyState of 4, which indicates the server
                   has responded (complete)
15             if (myRequest.readyState === 4) {
16                 // 5. insert the text sent by the server into the HTML of the '
                       ajax-content'
17                 document.getElementById('ajax-content').innerHTML =
                       myRequest.responseText;
18             }
```

```
19          };
20
21          function test(){
22              console.log('test');
23              var myRequest3 = new XMLHttpRequest();
24              myRequest3.open('POST', 'http://localhost:8080/test',true );
25              myRequest3.setRequestHeader('Content-Type','application/json');
26              myRequest3.onreadystatechange = function () {
27                  if (myRequest3.readyState === 4){// && myRequest2.status === 200) {
28                      console.log("ok request 3");
29                      console.log(myRequest3.getAllResponseHeaders());
30                      var json = JSON.parse(myRequest3.responseText);
31                      var nameNetwork = json.network[0].name; //mkyong
32                      var dataGenes = json.network[0].data;
33                      for(datakey in dataGenes){
34                          var node = dataGenes[datakey];
35                          console.log(node);
36                      }
37                      document.getElementById('nameNetwork').innerHTML = nameNetwork;
38                      document.getElementById('dataGenes').innerHTML = dataGenes;
39                  }
40              }
41              var toSendText ={"network": [{"name":"nameGiven","borneMax":100,"borneMin":0,"
                    borneEffectOnOthers":10,"borneEffectOnSelf":2,"steps":5,"nodes":[{"label":
                    "geneB", "type":"node"},{"label":"geneA", "type":"node"},{"label":"geneC",
                     "type":"node"}],"edges": [{"id":0, "from":"geneB", "to":"geneA", "effect"
                    :0},{"id":0, "from":"geneA", "to":"geneB", "effect":3}],"data": [{"node":"
                    geneB", "step":1, "niveau":2},{"node":"geneA", "step":0, "niveau":1}]}]};
42              var toSend = JSON.stringify(toSendText);
43              var data_test = JSON.stringify({"name": "test_geot"});
44              myRequest3.send(toSend);
45          }
46          function api() {
47              var genesList = [];
48              var dataPoints = [];
49              var matrix = [];
50              Matrices = [];
51              //using the function:
52              removeOptionsSelect();
53              var myRequest4 = new XMLHttpRequest();
54              myRequest4.open('POST', 'http://localhost:8080/api',true );
55              myRequest4.setRequestHeader('Content-Type','application/json');
56              myRequest4.onreadystatechange = function () {
57                  if (myRequest4.readyState === 4 && myRequest4.status === 201) {
58
59                      console.log(myRequest4.getAllResponseHeaders());
60
61                      var json = JSON.parse(myRequest4.responseText);
62                      Matrices = json.network;
63
64                      for (var j = 0; j < Matrices.length; j++) {
65                          // do something
66                          AddItemSelect(j,j);
67                      }
68
69                      //console.log(Matrices);
70
71                      var nameNetwork = json.network[0].name; //mkyong
72                      var steps = json.network[0].steps;
73                      var dataGenes = json.network[0].data;
74                      var edgeGenes = json.network[0].edges;
75                      var nodeGenes = json.network[0].nodes;
76                      // on rÃ©cupÃ¨re les arcs
77                      for(edgekey in edgeGenes){
78                          var node = edgeGenes[edgekey];
79                      }
80                      // on rÃ©cupÃ¨re les "genes"
81                      for(nodekey in nodeGenes){
82                          var node = nodeGenes[nodekey];
83                          genesList.push(node.label);
84                      }
85                      var nbreGenes = nodeGenes.length;
86
87                      // on rÃ©cupÃ¨re les donnÃ©es pour chaque step calculÃ©
88                      for(datakey in dataGenes){
89                          var node = dataGenes[datakey];
```

```
90                          //console.log(node);
91                          //console.log(node.step);
92                          //console.log(node.niveau);
93                      }
94                      addData(dataGenes,genesList,matrix);
95
96                      //matrix.forEach(printSimple);
97                      var chart = new CanvasJS.Chart("chartContainer", {
98                          title: {
99                              text: nameNetwork
100                         },
101                         axisX: {
102                         },
103                         axisY2: {
104
105                         },
106                         data: [
107                         ]
108                     });
109                     function addSerie(element){
110                         var newSeries = {
111                             showInLegend: true,
112                             legendText:element[0].name,
113                             type: "line",
114                             dataPoints: element
115                         };
116                         chart.options.data.push(newSeries);
117                     }
118                     matrix.forEach(addSerie);
119                     chart.render();
120                     document.getElementById('jsonCalcul').innerHTML = JSON.stringify(json,
                            undefined, 2);
121                     document.getElementById("button_tab_results").click()
122
123                 }
124                 if (myRequest4.readyState === 4 && myRequest4.status === 400) {
125                     var json = JSON.parse(myRequest4.responseText);
126                     document.getElementById('nameNetwork').innerHTML = 'erreur 400',
127                         document.getElementById('jsonCalcul').innerHTML = JSON.stringify(
                                json,undefined, 2);
128                 }
129
130
131
132             }
133
134
135             var toSendText = document.getElementById('jsonInput_text').value;
136             console.log(toSendText);
137             var toSend = JSON.stringify(toSendText);
138             myRequest4.send(toSendText);
139             //chart.render();
140             function addData(dataGenes, genesList, matrix) {
141                 console.log('addData');
142                 //Initialisation de la matrice
143                 for(var i=0; i<genesList.length; i++) {
144                     console.log("Initialisation de la matrice");
145                     console.log("i - matrix index = " + i);
146                     matrix[i] = [];
147                 }
148                 // on parcourt toutes les data renvoyÃ©es
149                 for (var i = 0; i < dataGenes.length; i++) {
150                     // on rÃ©cupÃ¨re l'indice du gÃªne dans la liste de genes
151                     var a = genesList.indexOf(dataGenes[i].node);
152                     //console.log(a);
153                     dataPoints.push({
154                         x: dataGenes[i].step,
155                         y: dataGenes[i].niveau
156                     });
157                     matrix[a].push({
158                         x: dataGenes[i].step,
159                         y: dataGenes[i].niveau,
160                         name:dataGenes[i].node
161                     });
162                 }
163                 //chart.render();
```

```
164             }
165             function printOut(element){
166                 console.log("name(x,y) = "+ element.name +"(" + element.x + ',' +
                        element.y + ')');
167             }
168             function printSimple(element){
169                 console.log(element);
170             }
171         }
172
173
174         function updateGraph(i){
175             var genesList = [];
176             var dataPoints = [];
177             var matrix = [];
178
179             var nameNetwork = Matrices[i].name; //mkyong
180             var steps = Matrices[i].steps;
181             var dataGenes = Matrices[i].data;
182             var edgeGenes = Matrices[i].edges;
183             var nodeGenes = Matrices[i].nodes;
184             // on rÃ©cupÃ¨re les arcs
185             for(edgekey in edgeGenes){
186                 var node = edgeGenes[edgekey];
187             }
188             // on rÃ©cupÃ¨re les "genes"
189             for(nodekey in nodeGenes){
190                 var node = nodeGenes[nodekey];
191                 genesList.push(node.label);
192             }
193             // on rÃ©cupÃ¨re les donnÃ©es pour chaque step calculÃ©
194             for(datakey in dataGenes){
195                 var node = dataGenes[datakey];
196             }
197             addData(dataGenes,genesList,matrix);
198             var chart = new CanvasJS.Chart("chartContainer", {
199                 title: {
200                     text: nameNetwork
201                 },
202                 axisX: {
203                 },
204                 axisY2: {
205
206                 },
207                 data: [
208
209                 ]
210             });
211             function addSerie(element){
212                 var newSeries = {
213                     showInLegend: true,
214                     legendText:element[0].name,
215                     type: "line",
216                     dataPoints: element
217                 };
218                 chart.options.data.push(newSeries);
219             }
220             matrix.forEach(addSerie);
221             chart.render();
222
223             function addData(dataGenes, genesList, matrix) {
224                 //Initialisation de la matrice
225                 for(var i=0; i<genesList.length; i++) {
226                     console.log("Initialisation de la matrice");
227                     console.log("i - matrix index = " + i);
228                     matrix[i] = [];
229                 }
230                 // on parcourt toutes les data renvoyÃ©es
231                 for (var i = 0; i < dataGenes.length; i++) {
232                     // on rÃ©cupÃ¨re l'indice du gÃªne dans la liste de genes
233                     var a = genesList.indexOf(dataGenes[i].node);
234                     //console.log(a);
235                     dataPoints.push({
236                         x: dataGenes[i].step,
237                         y: dataGenes[i].niveau
238                     });
```

```
239                         matrix[a].push({
240                             x: dataGenes[i].step,
241                             y: dataGenes[i].niveau,
242                             name:dataGenes[i].node
243                         });
244                     }
245                     //chart.render();
246                 }
247 
248                 document.getElementById('jsonCalcul').innerHTML = JSON.stringify(Matrices[i],
                        undefined,2);
249             }
250 
251             function printValue(){
252                 var jobValue = document.getElementById('jsonInput_text').value;
253                 console.log(jobValue);
254             }
255 
256             // add an item to the dropdown list
257             function AddItemSelect(text,value)
258             {
259                 // Create an Option object
260                 var opt = document.createElement("option");
261 
262                 // Assign text and value to Option object
263                 opt.text = "Solution #"+ text;
264                 opt.value = value;
265 
266                 // Add an Option object to Drop Down List Box
267                 document.getElementById('selection_answer').options.add(opt);
268             }
269 
270             function removeOptionsSelect()
271             {
272                 var selectbox = document.getElementById('selection_answer');
273                 var i;
274                 for(i = selectbox.options.length - 1 ; i >= 0 ; i--)
275                 {
276                     selectbox.remove(i);
277                 }
278             }
279 
280 
281 
282             function openTab(evt, tabName) {
283                 // Declare all variables
284                 var i, tabcontent, tablinks;
285 
286                 // Get all elements with class="tabcontent" and hide them
287                 tabcontent = document.getElementsByClassName("tabcontent");
288                 for (i = 0; i < tabcontent.length; i++) {
289                     tabcontent[i].style.display = "none";
290                 }
291 
292                 // Get all elements with class="tablinks" and remove the class "active"
293                 tablinks = document.getElementsByClassName("tablinks");
294                 for (i = 0; i < tablinks.length; i++) {
295                     tablinks[i].className = tablinks[i].className.replace(" active", "");
296                 }
297 
298                 // Show the current tab, and add an "active" class to the button that opened
                        the tab
299                 document.getElementById(tabName).style.display = "block";
300                 evt.currentTarget.className += " active";
301             }
302 
303 
304 
305         </script>
306     </head>
307 
308     <body>
309 
310     <h1>Reasoning on Gene Regulatory Networks</h1>
311 
312     <div class="tab">
```

```
313     <button class="tablinks" onclick="openTab(event, 'tab_inputs')" id="button_tab_default
            ">Inputs</button>
314     <button class="tablinks" onclick="openTab(event, 'tab_results')" id="
            button_tab_results">Results</button>
315     <!--<button class="tablinks" onclick="openTab(event, 'tab_results2')">NA</button>-->
316     <button class="tablinks" onclick="openTab(event, 'tab_help')">Help</button>
317 </div>
318
319 <!-- Tab content -->
320 <div id="tab_inputs" class="tabcontent">
321     <h3>Inputs</h3>
322     <p>Tab to declare inputs </p>
323     <button class="myButton" id="API" onclick="api()">Process</button>
324     <div id="jsonInput" align="center">
325         <p align="center">Input</p>
326         <textarea class="inline-txtarea"
327                   id="jsonInput_text"
328                   rows="50" cols="150"></textarea>
329     </div>
330 </div>
331
332 <div id="tab_results" class="tabcontent">
333     <h3>Results</h3>
334     <p>Tab to display the results of the processing</p>
335     <select id="selection_answer" name="solList" ></select>
336     <button id="updateGraph" onclick="updateGraph(document.getElementById('
            selection_answer').value)">Update</button>
337
338     <div class="container_result">
339             <div id="chartContainer" >
340                 <p align="center">Chart</p></div>
341             <div id="jsonOutput"><p align="center"></p>
342                 <h3 align="center">JSON Response</h3>
343                 <pre  class="inline-txtarea" id="jsonCalcul"></pre>
344             </div>
345         </div>
346 </div>
347
348 <div id="tab_results2" class="tabcontent">
349     <h3>Visual Graph</h3>
350     <p>Under Construction</p>
351 </div>
352 <div id="tab_help" class="tabcontent">
353     <h3>Help</h3>
354     <p>Example of correct JSON input file</p>
355     <pre id="jsonHelp"></pre>
356 </div>
357
358
359
360 <a href="network.html">network graph builder</a>
361
362
363 <script>
364     var jsonHelpfile={"network":[
365             {"name":"Lac Operon",
366                 "borneMax":10,
367                 "borneMin":0,
368                 "borneEffectOnOthers":3,
369                 "borneEffectOnSelf":0,
370                 "globalThreshold":2,
371                 "steps":7,
372                 "method":"tmp",
373                 "sparsity":0,
374                 "labeling":"all_ok",
375                 "nSol":10,
376                 "nodes":[
377                     {"label":"operonAYZ","type":"node"},
378                     {"label":"lactose","type":"node"},
379                     {"label":"lacI","type":"node"},
380                     {"label":"cap_cAMP","type":"node"},
381                     {"label":"glucose","type":"node"},
382                     {"label":"supportLacI","type":"control"}],
383                 "edges":[
384                     {"id":"toChange","from":"lactose","to":"lacI","threshold":"/","
                            borneEffect":"/","delay":0,"effect":-2},
```

```
                {"id":"toChange","from":"lacI","to":"operonAYZ","threshold":"/","
                    borneEffect":"/","delay":0,"effect":-3},
                {"id":"toChange","from":"operonAYZ","to":"glucose","threshold":"/","
                    borneEffect":"/","delay":0,"effect":1},
                {"id":"toChange","from":"operonAYZ","to":"lactose","threshold":"/","
                    borneEffect":"/","delay":0,"effect":-2},
                {"id":"toChange","from":"cap_cAMP","to":"operonAYZ","threshold":"/","
                    borneEffect":"/","delay":0,"effect":2},

                {"id":"toChange","from":"glucose","to":"cap_cAMP","threshold":"/","
                    borneEffect":"/","delay":0,"effect":-2},
                {"id":"toChange","from":"supportLacI","to":"lacI","threshold":"/","
                    borneEffect":"/","delay":0,"effect":1},

                {"id":"toChange","from":"glucose","to":"lacI","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"glucose","to":"lactose","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"glucose","to":"operonAYZ","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"operonAYZ","to":"lacI","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},

                {"id":"toChange","from":"operonAYZ","to":"cap_cAMP","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"lactose","to":"operonAYZ","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"lacI","to":"cap_cAMP","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"lacI","to":"lactose","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"lacI","to":"glucose","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},

                {"id":"toChange","from":"lactose","to":"cap_cAMP","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"lactose","to":"glucose","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"cap_cAMP","to":"lacI","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"cap_cAMP","to":"glucose","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"cap_cAMP","to":"lactose","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},

                {"id":"toChange","from":"supportLacI","to":"lactose","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"supportLacI","to":"glucose","threshold":"/","
                    borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"supportLacI","to":"cap_cAMP","threshold":"/",
                    "borneEffect":1,"delay":0,"effect":0},
                {"id":"toChange","from":"supportLacI","to":"operonAYZ","threshold":"/"
                    ,"borneEffect":1,"delay":0,"effect":0}
            ],
            "data":[
                {"node":"operonAYZ","step":0,"niveau":7},
                {"node":"lacI","step":0,"niveau":2},
                {"node":"cap_cAMP","step":0,"niveau":8},
                {"node":"lactose","step":0,"niveau":9},
                {"node":"glucose","step":0,"niveau":1},

                {"node":"operonAYZ","step":3,"niveau":7},
                {"node":"lacI","step":3,"niveau":0},
                {"node":"cap_cAMP","step":3,"niveau":4},
                {"node":"lactose","step":3,"niveau":3},
                {"node":"glucose","step":3,"niveau":4},


                {"node":"supportLacI","step":0,"niveau":1},
                {"node":"supportLacI","step":1,"niveau":1},
                {"node":"supportLacI","step":2,"niveau":1},
                {"node":"supportLacI","step":3,"niveau":1},
                {"node":"supportLacI","step":4,"niveau":1},
                {"node":"supportLacI","step":5,"niveau":1},
                {"node":"supportLacI","step":6,"niveau":1},
                {"node":"supportLacI","step":7,"niveau":1}
```

```
437                     ]}]}
438         // Get the element with id="defaultOpen" and click on it
439         document.getElementById("button_tab_default").click();
440         document.getElementById("jsonHelp").innerHTML = JSON.stringify(jsonHelpfile,undefined,
                2);
441     </script>
442
443     <script src="https://canvasjs.com/assets/script/canvasjs.min.js">
444     </script>
445
446     </body>
447
448     </html>
```

Appendix B

Log files

```
network_test_memoisation_steps8_nodes3_edges4_data3.json,1534257926.621911,FAIL,0.0034012794494628906,[s]
network_test_tmp_steps4_nodes3_edges2_data3.json,1534257926.62534,timeout,inf,[s]
network_test_tmp_steps2_nodes4_edges4_data0.json,1534257931.626098,OK,0.008955001831054688,[s]
network_test_memoisation_steps6_nodes3_edges4_data2.json,1534257931.6350858,FAIL,0.002515554428100586,[s]
network_test_tmp_steps4_nodes3_edges4_data0.json,1534257931.6376176,FAIL,0.0035729408264160156,[s]
network_test_lineaire_steps2_nodes4_edges4_data1.json,1534257931.6412082,OK,0.010091543197631836,[s]
network_test_tmp_steps6_nodes3_edges2_data1.json,1534257931.651334,FAIL,3.0350427627563477,[s]
network_test_memoisation_steps8_nodes4_edges0_data0.json,1534257934.68642,timeout,inf,[s]
network_test_tmp_steps6_nodes4_edges0_data0.json,1534257939.686757,timeout,inf,[s]
network_test_tmp_steps4_nodes3_edges0_data1.json,1534257944.6875803,timeout,inf,[s]
network_test_lineaire_steps4_nodes4_edges4_data1.json,1534257949.6878636,timeout,inf,[s]
network_test_memoisation_steps2_nodes4_edges2_data2.json,1534257954.6881483,OK,0.01009058952331543,[s]
network_test_lineaire_steps4_nodes4_edges6_data0.json,1534257954.698291,OK,0.012897014617919922,[s]
network_test_tmp_steps2_nodes3_edges4_data0.json,1534257954.7112453,FAIL,0.0020837783813476562,[s]
network_test_tmp_steps4_nodes4_edges6_data3.json,1534257954.7133608,OK,0.05526876449584961,[s]
network_test_tmp_steps4_nodes3_edges2_data1.json,1534257954.768684,FAIL,0.8962094783782959,[s]
network_test_lineaire_steps4_nodes4_edges6_data1.json,1534257955.6649578,timeout,inf,[s]
network_test_lineaire_steps2_nodes3_edges0_data1.json,1534257960.6662986,OK,0.022145509719848633,[s]
network_test_lineaire_steps4_nodes2_edges0_data0.json,1534257960.6885002,FAIL,0.0026259422302246094,[s]
network_test_memoisation_steps2_nodes4_edges4_data2.json,1534257960.6911595,OK,0.010919570922851562,[s]
network_test_lineaire_steps4_nodes2_edges2_data1.json,1534257960.702098,FAIL,0.0036232471466064453,[s]
network_test_memoisation_steps4_nodes4_edges6_data1.json,1534257960.7057827,OK,0.015269517898559570,[s]
network_test_lineaire_steps4_nodes4_edges0_data3.json,1534257960.7210834,timeout,inf,[s]
network_test_memoisation_steps8_nodes2_edges2_data1.json,1534257965.7219267,FAIL,0.08501720428466797,[s]
network_test_tmp_steps6_nodes3_edges0_data0.json,1534257965.8070018,FAIL,0.023842573165893555,[s]
network_test_tmp_steps4_nodes4_edges2_data1.json,1534257965.8309007,OK,0.0231168270111084,[s]
network_test_lineaire_steps4_nodes2_edges2_data2.json,1534257965.8540723,FAIL,0.006420135498046875,[s]
network_test_lineaire_steps6_nodes3_edges0_data0.json,1534257965.860525,FAIL,0.030551433563232422,[s]
network_test_lineaire_steps6_nodes4_edges4_data2.json,1534257965.8911335,timeout,inf,[s]
network_test_tmp_steps8_nodes4_edges4_data3.json,1534257970.891368,timeout,inf,[s]
network_test_memoisation_steps8_nodes4_edges4_data3.json,1534257975.8916094,timeout,inf,[s]
network_test_lineaire_steps6_nodes3_edges4_data3.json,1534257980.8924885,FAIL,0.003543376922607422,[s]
network_test_lineaire_steps6_nodes4_edges0_data1.json,1534257980.896052,timeout,inf,[s]
network_test_memoisation_steps6_nodes3_edges0_data0.json,1534257985.8962789,FAIL,0.4291994571685791,[s]
network_test_tmp_steps6_nodes2_edges0_data0.json,1534257986.3255384,FAIL,0.002650022506713867,[s]
network_test_lineaire_steps6_nodes4_edges6_data3.json,1534257986.32822,timeout,inf,[s]
network_test_lineaire_steps2_nodes2_edges2_data0.json,1534257991.3284755,FAIL,0.0011882781982421875,[s]
network_test_lineaire_steps8_nodes4_edges0_data2.json,1534257991.3297167,timeout,inf,[s]
network_test_tmp_steps8_nodes2_edges2_data2.json,1534257996.3299377,FAIL,0.036932945251464844,[s]
network_test_tmp_steps8_nodes4_edges2_data1.json,1534257996.366989,timeout,inf,[s]
network_test_lineaire_steps6_nodes4_edges0_data2.json,1534258001.3672714,timeout,inf,[s]
network_test_tmp_steps8_nodes4_edges2_data0.json,1534258006.3678608,timeout,inf,[s]
network_test_tmp_steps6_nodes4_edges4_data1.json,1534258011.3688195,timeout,inf,[s]
network_test_memoisation_steps8_nodes3_edges0_data1.json,1534258016.3698738,timeout,inf,[s]
network_test_memoisation_steps2_nodes3_edges0_data3.json,1534258021.3701432,OK,0.005298614501953125,[s]
network_test_memoisation_steps6_nodes4_edges4_data3.json,1534258021.375476,timeout,inf,[s]
network_test_tmp_steps2_nodes3_edges2_data3.json,1534258026.3768625,OK,0.005632877349853516,[s]
network_test_memoisation_steps2_nodes3_edges4_data3.json,1534258026.3825307,FAIL,0.0013062953948974609,[s]
network_test_tmp_steps6_nodes4_edges4_data0.json,1534258026.3838506,OK,0.014627456665039062,[s]
network_test_tmp_steps6_nodes3_edges0_data1.json,1534258026.398517,timeout,inf,[s]
network_test_tmp_steps6_nodes2_edges2_data2.json,1534258031.3993976,FAIL,0.014746189117431641,[s]
network_test_tmp_steps6_nodes2_edges0_data2.json,1534258031.4141874,FAIL,0.07350373268127441,[s]
network_test_memoisation_steps6_nodes2_edges2_data1.json,1534258031.4877608,FAIL,0.022589206695556641,[s]
network_test_lineaire_steps2_nodes4_edges2_data1.json,1534258031.5104218,OK,0.009817838668823242,[s]
network_test_lineaire_steps2_nodes3_edges0_data3.json,1534258031.5202742,OK,0.004586696624755859,[s]
network_test_tmp_steps8_nodes3_edges0_data3.json,1534258031.5248783,timeout,inf,[s]
network_test_lineaire_steps4_nodes3_edges2_data0.json,1534258036.5252936,FAIL,0.018312692642211914,[s]
network_test_lineaire_steps2_nodes4_edges6_data3.json,1534258036.5436509,OK,0.007188796997070312,[s]
network_test_tmp_steps6_nodes4_edges0_data1.json,1534258036.5508928,timeout,inf,[s]
network_test_memoisation_steps4_nodes4_edges4_data3.json,1534258041.5511918,OK,0.059702873229980469,[s]
network_test_lineaire_steps6_nodes2_edges2_data3.json,1534258041.6109295,FAIL,0.023845672607421875,[s]
network_test_tmp_steps2_nodes3_edges4_data2.json,1534258041.6348133,FAIL,0.0013659000396728516,[s]
network_test_memoisation_steps6_nodes4_edges2_data0.json,1534258041.6361933,timeout,inf,[s]
network_test_memoisation_steps6_nodes4_edges2_data3.json,1534258046.6373925,timeout,inf,[s]
network_test_lineaire_steps2_nodes3_edges0_data2.json,1534258051.6377418,OK,0.0035724639892578125,[s]
network_test_tmp_steps2_nodes3_edges2_data0.json,1534258051.641329,FAIL,0.009250640869140625,[s]
network_test_tmp_steps6_nodes4_edges6_data0.json,1534258051.6505983,OK,0.013431787490844727,[s]
network_test_lineaire_steps8_nodes4_edges0_data1.json,1534258051.6640842,timeout,inf,[s]
network_test_lineaire_steps6_nodes3_edges0_data3.json,1534258056.6648543,timeout,inf,[s]
network_test_memoisation_steps8_nodes2_edges0_data1.json,1534258061.6659956,FAIL,0.7156186103820801,[s]
network_test_lineaire_steps4_nodes2_edges0_data3.json,1534258062.381653,OK,0.04151940345764160,[s]
network_test_lineaire_steps6_nodes4_edges0_data3.json,1534258062.4231896,timeout,inf,[s]
network_test_lineaire_steps8_nodes4_edges2_data1.json,1534258067.4242547,timeout,inf,[s]
network_test_tmp_steps2_nodes4_edges2_data3.json,1534258072.4250498,OK,0.008034944534301758,[s]
network_test_lineaire_steps2_nodes2_edges0_data0.json,1534258072.4331155,FAIL,0.0017614364624023438,[s]
network_test_lineaire_steps2_nodes4_edges0_data2.json,1534258072.4349086,OK,0.010200262069702148,[s]
network_test_memoisation_steps4_nodes3_edges4_data3.json,1534258072.44515,FAIL,0.0024793148040771484,[s]
```

```
78  network_test_tmp_steps8_nodes4_edges0_data0.json,1534258072.4476898,timeout,inf,[s]
79  network_test_lineaire_steps6_nodes2_edges2_data1.json,1534258077.4480188,FAIL,0.008008480072021484,[s]
80  network_test_memoisation_steps4_nodes4_edges2_data1.json,1534258077.4560835,OK,0.017900705337524414,[s]
81  network_test_memoisation_steps6_nodes3_edges4_data1.json,1534258077.4740343,FAIL,0.003706455230712890 6,[s]
82  network_test_tmp_steps4_nodes3_edges0_data0.json,1534258077.4778445,FAIL,0.023485660552978516,[s]
83  network_test_memoisation_steps8_nodes3_edges2_data2.json,1534258077.5013855,timeout,inf,[s]
84  network_test_tmp_steps4_nodes2_edges2_data3.json,1534258082.5029202,FAIL,0.01533961296081543,[s]
85  network_test_tmp_steps4_nodes2_edges2_data2.json,1534258082.5182893,FAIL,0.007619619369506836,[s]
86  network_test_tmp_steps6_nodes4_edges6_data2.json,1534258082.525942,timeout,inf,[s]
87  network_test_memoisation_steps4_nodes3_edges4_data1.json,1534258087.5262651,FAIL,0.003742218017578125,[s]
88  network_test_lineaire_steps8_nodes3_edges0_data1.json,1534258087.5300415,timeout,inf,[s]
89  network_test_tmp_steps2_nodes2_edges0_data1.json,1534258092.530307,OK,0.002141237258911133,[s]
90  network_test_memoisation_steps6_nodes3_edges0_data2.json,1534258092.5324612,timeout,inf,[s]
91  network_test_memoisation_steps8_nodes4_edges0_data1.json,1534258097.5326736,timeout,inf,[s]
92  network_test_lineaire_steps2_nodes4_edges4_data3.json,1534258102.5328918,OK,0.008754253387451172,[s]
93  network_test_lineaire_steps6_nodes4_edges2_data1.json,1534258102.5416613,OK,0.05423140525817871,[s]
94  network_test_tmp_steps8_nodes3_edges0_data1.json,1534258102.5959275,timeout,inf,[s]
95  network_test_lineaire_steps6_nodes4_edges2_data2.json,1534258107.5968828,timeout,inf,[s]
96  network_test_memoisation_steps8_nodes3_edges4_data2.json,1534258112.5971887,FAIL,0.003836393356323242,[s]
97  network_test_memoisation_steps6_nodes4_edges6_data1.json,1534258112.6010394,timeout,inf,[s]
98  network_test_lineaire_steps8_nodes4_edges4_data1.json,1534258117.6024446,timeout,inf,[s]
99  network_test_memoisation_steps4_nodes3_edges4_data2.json,1534258122.6027272,FAIL,0.002170324325561523 4,[s]
100 network_test_tmp_steps8_nodes4_edges6_data0.json,1534258122.6049302,OK,0.0165319442749023 44,[s]
101 network_test_tmp_steps4_nodes3_edges0_data2.json,1534258122.6215012,OK,0.015901565551757812,[s]
102 network_test_lineaire_steps4_nodes3_edges0_data2.json,1534258122.6374354,OK,1.8130903244018555,[s]
103 network_test_memoisation_steps4_nodes3_edges2_data1.json,1534258124.450562,timeout,inf,[s]
104 network_test_tmp_steps4_nodes3_edges4_data3.json,1534258129.4512882,FAIL,0.001718044281005859 4,[s]
105 network_test_lineaire_steps2_nodes4_edges0_data3.json,1534258129.4530213,OK,0.007387876510620117,[s]
106 network_test_tmp_steps2_nodes2_edges2_data3.json,1534258129.4604409,OK,0.001503229141235351 6,[s]
107 network_test_lineaire_steps8_nodes2_edges0_data3.json,1534258129.4619582,FAIL,0.4006962776184082,[s]
108 network_test_memoisation_steps4_nodes4_edges0_data2.json,1534258129.8626914,OK,0.016326665878295 9,[s]
109 network_test_tmp_steps2_nodes4_edges6_data1.json,1534258129.8790693,OK,0.010194063186645508,[s]
110 network_test_tmp_steps4_nodes4_edges4_data1.json,1534258129.889281,OK,0.02176809310913086,[s]
111 network_test_memoisation_steps2_nodes4_edges2_data1.json,1534258129.911085,OK,0.008264064788818 36,[s]
112 network_test_memoisation_steps6_nodes4_edges0_data2.json,1534258129.9193628,timeout,inf,[s]
113 network_test_tmp_steps4_nodes3_edges4_data2.json,1534258134.9196343,FAIL,0.001907825469970703 1,[s]
114 network_test_lineaire_steps2_nodes4_edges2_data3.json,1534258134.921571,OK,0.007441997528076172,[s]
115 network_test_tmp_steps2_nodes3_edges0_data2.json,1534258134.9290447,OK,0.005779981613159 18,[s]
116 network_test_memoisation_steps2_nodes3_edges2_data1.json,1534258134.934841,OK,0.004605770111083984,[s]
117 network_test_tmp_steps2_nodes2_edges2_data1.json,1534258134.9394891,FAIL,0.002759933471679687 5,[s]
118 network_test_tmp_steps8_nodes3_edges4_data3.json,1534258134.9422638,FAIL,0.002164125442504883,[s]
119 network_test_tmp_steps4_nodes4_edges6_data1.json,1534258134.9444413,OK,0.018872737884521484,[s]
120 network_test_memoisation_steps4_nodes4_edges0_data3.json,1534258134.96335,OK,1.039665222167968 8,[s]
121 network_test_lineaire_steps8_nodes2_edges2_data0.json,1534258136.0030358,FAIL,0.002927780151367187 5,[s]
122 network_test_memoisation_steps8_nodes4_edges4_data0.json,1534258136.0059972,OK,0.017101049423217773,[s]
123 network_test_memoisation_steps2_nodes4_edges4_data3.json,1534258136.0231187,OK,0.007961034774780273,[s]
124 network_test_memoisation_steps6_nodes4_edges4_data2.json,1534258136.0311189,timeout,inf,[s]
125 network_test_tmp_steps6_nodes4_edges6_data1.json,1534258141.0314343,timeout,inf,[s]
126 network_test_lineaire_steps8_nodes4_edges4_data0.json,1534258146.0317645,OK,0.039307355880737305,[s]
127 network_test_tmp_steps8_nodes2_edges0_data0.json,1534258146.071091,FAIL,0.005718469619750976 6,[s]
128 network_test_lineaire_steps6_nodes3_edges0_data2.json,1534258146.0768425,timeout,inf,[s]
129 network_test_memoisation_steps2_nodes3_edges2_data2.json,1534258151.0775702,OK,0.003670692244384765 62,[s]
130 network_test_lineaire_steps6_nodes2_edges0_data3.json,1534258151.0812912,OK,0.144830465316772 46,[s]
131 network_test_memoisation_steps4_nodes4_edges2_data3.json,1534258151.2261786,OK,0.049218893051147 46,[s]
132 network_test_memoisation_steps2_nodes4_edges6_data1.json,1534258151.2754526,OK,0.014240741729736328,[s]
133 network_test_memoisation_steps6_nodes4_edges6_data3.json,1534258151.2897763,timeout,inf,[s]
134 network_test_memoisation_steps8_nodes2_edges0_data0.json,1534258156.2911348,FAIL,0.019431114196777344,[s]
135 network_test_memoisation_steps4_nodes2_edges0_data2.json,1534258156.3106325,OK,0.003917455673217773 4,[s]
136 network_test_tmp_steps4_nodes2_edges0_data2.json,1534258156.3145819,OK,0.005607843399047851 6,[s]
137 network_test_memoisation_steps4_nodes4_edges4_data1.json,1534258156.3203871,OK,0.020006566467285 16,[s]
138 network_test_tmp_steps4_nodes4_edges0_data3.json,1534258156.340447,timeout,inf,[s]
139 network_test_lineaire_steps4_nodes4_edges6_data2.json,1534258161.341478,timeout,inf,[s]
140 network_test_memoisation_steps8_nodes4_edges6_data2.json,1534258166.3423893,timeout,inf,[s]
141 network_test_lineaire_steps2_nodes3_edges4_data0.json,1534258171.3427572,FAIL,0.003376723358154297,[s]
142 network_test_memoisation_steps2_nodes4_edges6_data2.json,1534258171.346155,OK,0.007865428924560547,[s]
143 network_test_memoisation_steps4_nodes3_edges4_data0.json,1534258171.3540564,FAIL,0.003133773803710937 5,[s]
144 network_test_tmp_steps2_nodes4_edges0_data0.json,1534258171.3572516,OK,0.008712768554687 5,[s]
145 network_test_memoisation_steps4_nodes3_edges2_data3.json,1534258171.3660069,timeout,inf,[s]
146 network_test_memoisation_steps8_nodes3_edges2_data0.json,1534258176.3669522,FAIL,0.254175186157226 56,[s]
147 network_test_memoisation_steps4_nodes3_edges0_data1.json,1534258176.6211684,timeout,inf,[s]
148 network_test_lineaire_steps2_nodes4_edges2_data0.json,1534258181.622002,OK,0.009401559829711914,[s]
149 network_test_tmp_steps2_nodes4_edges0_data2.json,1534258181.6314378,OK,0.011676549911499023,[s]
150 network_test_lineaire_steps6_nodes3_edges2_data0.json,1534258181.6431568,FAIL,0.021211385726928 71,[s]
151 network_test_lineaire_steps2_nodes2_edges2_data2.json,1534258181.6644106,OK,0.001879215240478 5156,[s]
152 network_test_memoisation_steps8_nodes4_edges6_data1.json,1534258181.6663237,timeout,inf,[s]
153 network_test_lineaire_steps6_nodes2_edges0_data1.json,1534258186.667847,FAIL,0.021958112716674805,[s]
154 network_test_lineaire_steps4_nodes2_edges2_data3.json,1534258186.6898477,FAIL,0.008080482482910156,[s]
155 network_test_tmp_steps2_nodes4_edges6_data0.json,1534258186.6979527,OK,0.007429122924804687 5,[s]
156 network_test_memoisation_steps2_nodes2_edges2_data0.json,1534258186.7054234,FAIL,0.001536607742309 5703,[s]
157 network_test_lineaire_steps6_nodes4_edges2_data3.json,1534258186.7069788,timeout,inf,[s]
158 network_test_memoisation_steps6_nodes2_edges2_data2.json,1534258191.7071335,FAIL,0.086812734603881 84,[s]
159 network_test_memoisation_steps4_nodes3_edges0_data2.json,1534258191.7939875,OK,0.008085012435913086,[s]
160 network_test_tmp_steps8_nodes3_edges2_data0.json,1534258191.8021076,FAIL,0.027761459350585938,[s]
161 network_test_lineaire_steps6_nodes3_edges4_data1.json,1534258191.829933,FAIL,0.002533197402954101 6,[s]
162 network_test_memoisation_steps4_nodes3_edges2_data2.json,1534258191.8325067,OK,0.007577419281005859,[s]
163 network_test_tmp_steps8_nodes4_edges6_data2.json,1534258191.840118,timeout,inf,[s]
164 network_test_lineaire_steps8_nodes4_edges0_data3.json,1534258196.840307,timeout,inf,[s]
165 network_test_tmp_steps6_nodes2_edges2_data3.json,1534258201.8407862,FAIL,0.021468162536621094,[s]
166 network_test_tmp_steps2_nodes2_edges0_data3.json,1534258201.8622794,OK,0.002227306365966797,[s]
167 network_test_memoisation_steps4_nodes4_edges6_data0.json,1534258201.8645222,OK,0.010958433151245117,[s]
168 network_test_memoisation_steps6_nodes4_edges4_data1.json,1534258201.875508,timeout,inf,[s]
169 network_test_tmp_steps8_nodes4_edges2_data2.json,1534258206.8757324,timeout,inf,[s]
170 network_test_lineaire_steps8_nodes4_edges2_data2.json,1534258211.876111,timeout,inf,[s]
171 network_test_lineaire_steps2_nodes4_edges0_data1.json,1534258216.8771772,OK,0.008045911788940 43,[s]
172 network_test_memoisation_steps4_nodes2_edges2_data3.json,1534258216.885254,FAIL,0.031810760498046875,[s]
173 network_test_memoisation_steps2_nodes4_edges2_data3.json,1534258216.9171107,OK,0.007609844207763672,[s]
174 network_test_lineaire_steps6_nodes4_edges6_data0.json,1534258216.9247463,OK,0.014336585998535156,[s]
175 network_test_tmp_steps8_nodes3_edges0_data2.json,1534258216.9391258,timeout,inf,[s]
176 network_test_lineaire_steps8_nodes3_edges4_data1.json,1534258221.9400315,FAIL,0.002900838851928711,[s]
177 network_test_tmp_steps6_nodes3_edges4_data0.json,1534258221.9429643,FAIL,0.003169298171997070 3,[s]
178 network_test_tmp_steps4_nodes4_edges4_data3.json,1534258221.946165,OK,0.15393853187561035,[s]
179 network_test_memoisation_steps4_nodes4_edges0_data1.json,1534258222.1001427,OK,0.5282185077667236,[s]
180 network_test_tmp_steps2_nodes4_edges4_data1.json,1534258222.628416,OK,0.007724285125732422,[s]
```

```
network_test_memoisation_steps6_nodes3_edges2_data2.json,1534258222.6361716,timeout,inf,[s]
network_test_lineaire_steps2_nodes3_edges4_data3.json,1534258227.6373103,FAIL,0.001497030258178711,[s]
network_test_tmp_steps2_nodes4_edges0_data1.json,1534258227.6388426,OK,0.008683443069458008,[s]
network_test_memoisation_steps2_nodes3_edges2_data0.json,1534258227.6475585,FAIL,0.13155150413513184,[s]
network_test_lineaire_steps8_nodes4_edges2_data0.json,1534258227.7791672,OK,0.06592464447021484,[s]
network_test_tmp_steps8_nodes2_edges0_data3.json,1534258227.8451445,FAIL,0.35985445976257324,[s]
network_test_lineaire_steps2_nodes4_edges6_data2.json,1534258228.2050552,OK,0.006693124771118164,[s]
network_test_lineaire_steps6_nodes2_edges2_data0.json,1534258228.211781,FAIL,0.0019474641372680664,[s]
network_test_lineaire_steps4_nodes3_edges4_data1.json,1534258228.2137601,FAIL,0.001847982406616211,[s]
network_test_memoisation_steps2_nodes4_edges6_data0.json,1534258228.2156389,OK,0.008750200271606445,[s]
network_test_tmp_steps4_nodes4_edges4_data2.json,1534258228.224468,timeout,inf,[s]
network_test_tmp_steps6_nodes3_edges0_data3.json,1534258233.225474,timeout,inf,[s]
network_test_memoisation_steps8_nodes4_edges6_data3.json,1534258238.225769,timeout,inf,[s]
network_test_lineaire_steps8_nodes3_edges0_data3.json,1534258243.2266219,timeout,inf,[s]
network_test_memoisation_steps2_nodes4_edges6_data3.json,1534258248.2280803,OK,0.007083415985107422,[s]
network_test_tmp_steps2_nodes3_edges2_data1.json,1534258248.2351956,OK,0.004446506500244141,[s]
network_test_lineaire_steps8_nodes3_edges2_data0.json,1534258248.2396755,FAIL,0.03448820114135742,[s]
network_test_tmp_steps8_nodes4_edges4_data1.json,1534258248.2742205,timeout,inf,[s]
network_test_lineaire_steps8_nodes4_edges6_data2.json,1534258253.2745228,timeout,inf,[s]
network_test_tmp_steps6_nodes2_edges0_data3.json,1534258258.2754745,OK,0.13626933097839355,[s]
network_test_memoisation_steps8_nodes4_edges2_data0.json,1534258258.4117985,timeout,inf,[s]
network_test_memoisation_steps4_nodes4_edges0_data0.json,1534258263.4121394,timeout,inf,[s]
network_test_memoisation_steps8_nodes4_edges0_data3.json,1534258268.4124947,timeout,inf,[s]
network_test_tmp_steps6_nodes4_edges0_data2.json,1534258273.4146197,timeout,inf,[s]
network_test_memoisation_steps2_nodes2_edges0_data0.json,1534258278.4148972,FAIL,0.006427764892578125,[s]
network_test_memoisation_steps2_nodes4_edges0_data0.json,1534258278.4213593,OK,0.006208181381225586,[s]
network_test_lineaire_steps2_nodes4_edges4_data0.json,1534258278.427611,OK,0.010190486907958984,[s]
network_test_memoisation_steps6_nodes2_edges2_data3.json,1534258278.437843,FAIL,0.11266565322875977,[s]
network_test_memoisation_steps8_nodes2_edges0_data2.json,1534258278.55056,FAIL,4.516624927520752,[s]
network_test_lineaire_steps2_nodes3_edges0_data0.json,1534258283.0672102,FAIL,0.015125751495361328,[s]
network_test_tmp_steps4_nodes2_edges2_data1.json,1534258283.0823581,FAIL,0.00348663330078125,[s]
network_test_memoisation_steps6_nodes2_edges0_data3.json,1534258283.0858643,OK,0.057564735412597656,[s]
network_test_lineaire_steps4_nodes4_edges2_data1.json,1534258283.1434824,OK,0.11573052406311035,[s]
network_test_tmp_steps4_nodes4_edges2_data2.json,1534258283.2592487,OK,0.0189614295959472 66,[s]
network_test_memoisation_steps8_nodes4_edges0_data2.json,1534258283.2782505,timeout,inf,[s]
network_test_tmp_steps6_nodes3_edges4_data3.json,1534258288.2802076,FAIL,0.0019192695617675781,[s]
network_test_tmp_steps6_nodes4_edges2_data1.json,1534258288.282142,timeout,inf,[s]
network_test_memoisation_steps8_nodes3_edges2_data3.json,1534258293.2833023,timeout,inf,[s]
network_test_memoisation_steps8_nodes4_edges6_data0.json,1534258298.2848775,OK,0.011673450469970703,[s]
network_test_lineaire_steps8_nodes2_edges2_data3.json,1534258298.296571,FAIL,0.07785058021545 41,[s]
network_test_lineaire_steps2_nodes3_edges4_data1.json,1534258298.3744771,FAIL,0.0015988349914550781,[s]
network_test_tmp_steps8_nodes3_edges4_data0.json,1534258298.376108,FAIL,0.003931283950805664,[s]
network_test_memoisation_steps4_nodes4_edges2_data0.json,1534258298.38007,timeout,inf,[s]
network_test_memoisation_steps2_nodes4_edges0_data2.json,1534258303.3815005,OK,0.009040594100952148,[s]
network_test_tmp_steps6_nodes4_edges2_data3.json,1534258303.3905957,timeout,inf,[s]
network_test_tmp_steps6_nodes3_edges2_data2.json,1534258308.3919954,timeout,inf,[s]
network_test_lineaire_steps4_nodes4_edges0_data2.json,1534258313.3929992,OK,0.055064678192138 67,[s]
network_test_tmp_steps2_nodes3_edges2_data2.json,1534258313.448101,OK,0.006190776824951172,[s]
network_test_memoisation_steps8_nodes2_edges2_data2.json,1534258313.4545898,FAIL,0.3160541057586 67,[s]
network_test_memoisation_steps6_nodes4_edges2_data1.json,1534258313.7707148,timeout,inf,[s]
network_test_tmp_steps4_nodes3_edges0_data3.json,1534258318.7723825,timeout,inf,[s]
network_test_tmp_steps2_nodes3_edges0_data1.json,1534258323.7753832,OK,0.0044646263122558 59,[s]
network_test_memoisation_steps6_nodes3_edges2_data0.json,1534258323.7798867,FAIL,0.17276787757873535,[s]
network_test_lineaire_steps8_nodes2_edges0_data1.json,1534258323.9527154,FAIL,0.0581052303314209,[s]
network_test_tmp_steps6_nodes2_edges0_data1.json,1534258324.010861,FAIL,0.018726110458374023,[s]
network_test_lineaire_steps8_nodes4_edges4_data2.json,1534258324.0296476,timeout,inf,[s]
network_test_memoisation_steps8_nodes4_edges4_data1.json,1534258329.0305119,timeout,inf,[s]
network_test_tmp_steps8_nodes4_edges0_data1.json,1534258334.0307093,timeout,inf,[s]
network_test_tmp_steps4_nodes3_edges2_data0.json,1534258339.0325792,FAIL,0.013915538787841797,[s]
network_test_tmp_steps2_nodes3_edges4_data1.json,1534258339.046524,FAIL,0.0017409324645996094,[s]
network_test_lineaire_steps8_nodes3_edges4_data3.json,1534258339.0482886,FAIL,0.002395629882812 5,[s]
network_test_tmp_steps6_nodes3_edges2_data3.json,1534258339.0507035,timeout,inf,[s]
network_test_lineaire_steps4_nodes4_edges6_data3.json,1534258344.0518494,timeout,inf,[s]
network_test_memoisation_steps6_nodes4_edges0_data1.json,1534258349.0530443,timeout,inf,[s]
network_test_lineaire_steps8_nodes2_edges2_data2.json,1534258354.054592,FAIL,0.044090509414672 85,[s]
network_test_lineaire_steps2_nodes2_edges0_data1.json,1534258354.0987444,OK,0.004397869110107422,[s]
network_test_memoisation_steps2_nodes3_edges0_data1.json,1534258354.1031754,OK,0.0036540031433105 47,[s]
network_test_tmp_steps4_nodes3_edges2_data2.json,1534258354.106862,OK,0.010120391845703125,[s]
network_test_tmp_steps8_nodes2_edges2_data0.json,1534258354.1170254,FAIL,0.0035331249237060547,[s]
network_test_memoisation_steps8_nodes3_edges0_data2.json,1534258354.1205862,timeout,inf,[s]
network_test_lineaire_steps6_nodes3_edges4_data2.json,1534258359.1223793,FAIL,0.003838777542114258,[s]
network_test_tmp_steps2_nodes4_edges2_data1.json,1534258359.1262558,OK,0.007694959640502 93,[s]
network_test_tmp_steps8_nodes4_edges0_data3.json,1534258359.1339653,timeout,inf,[s]
network_test_lineaire_steps6_nodes2_edges0_data0.json,1534258364.135407,FAIL,0.0050199031829833984,[s]
network_test_memoisation_steps4_nodes4_edges6_data3.json,1534258364.1405072,OK,0.036220788955688 48,[s]
network_test_memoisation_steps8_nodes4_edges2_data2.json,1534258364.1767454,timeout,inf,[s]
network_test_memoisation_steps8_nodes2_edges0_data3.json,1534258369.1770954,timeout,inf,[s]
network_test_lineaire_steps8_nodes3_edges0_data0.json,1534258374.1780198,FAIL,0.050466775894165 04,[s]
network_test_tmp_steps6_nodes2_edges2_data0.json,1534258374.2285278,FAIL,0.0016424655914306 64,[s]
network_test_memoisation_steps4_nodes3_edges0_data0.json,1534258374.2302024,FAIL,0.5960931777954102,[s]
network_test_tmp_steps2_nodes2_edges2_data2.json,1534258374.8263533,OK,0.0015540122985839844,[s]
network_test_memoisation_steps8_nodes3_edges0_data3.json,1534258374.8279383,timeout,inf,[s]
network_test_memoisation_steps2_nodes2_edges0_data3.json,1534258379.8295279,OK,0.003924131393432617,[s]
network_test_lineaire_steps6_nodes4_edges6_data2.json,1534258379.8335106,timeout,inf,[s]
network_test_tmp_steps8_nodes2_edges0_data1.json,1534258384.8344767,FAIL,0.0535287857055664 06,[s]
network_test_memoisation_steps4_nodes3_edges2_data0.json,1534258384.8880265,FAIL,0.2057371139526367 2,[s]
network_test_tmp_steps2_nodes4_edges4_data3.json,1534258385.093816,OK,0.006990671157836914,[s]
network_test_lineaire_steps8_nodes3_edges4_data2.json,1534258385.100838,FAIL,0.0021095275878906 25,[s]
network_test_tmp_steps8_nodes3_edges2_data3.json,1534258385.102979,timeout,inf,[s]
network_test_tmp_steps4_nodes2_edges0_data0.json,1534258390.1041014,FAIL,0.003674030303955078,[s]
network_test_memoisation_steps2_nodes4_edges0_data1.json,1534258390.1077929,OK,0.007708787918090 82,[s]
network_test_lineaire_steps4_nodes3_edges2_data1.json,1534258390.115538,FAIL,1.0373456478118896,[s]
network_test_lineaire_steps8_nodes2_edges2_data1.json,1534258391.1529384,FAIL,0.018914699554443 36,[s]
network_test_memoisation_steps4_nodes2_edges2_data0.json,1534258391.171935,FAIL,0.0018496513366699219,[s]
network_test_lineaire_steps4_nodes2_edges0_data2.json,1534258391.17382,OK,0.021175861358642578,[s]
network_test_lineaire_steps4_nodes4_edges2_data2.json,1534258391.1950457,timeout,inf,[s]
network_test_lineaire_steps6_nodes2_edges0_data2.json,1534258396.1958892,FAIL,0.0784184932708740 2,[s]
network_test_lineaire_steps4_nodes3_edges0_data3.json,1534258396.2743397,OK,1.6214630603790283,[s]
network_test_tmp_steps2_nodes4_edges4_data2.json,1534258397.8958383,OK,0.007900953292846 68,[s]
network_test_memoisation_steps2_nodes2_edges0_data1.json,1534258397.9037533,OK,0.0018653869628906 25,[s]
network_test_lineaire_steps8_nodes3_edges2_data1.json,1534258397.9056313,timeout,inf,[s]
network_test_lineaire_steps4_nodes3_edges4_data0.json,1534258402.906584,FAIL,0.0023491382598876953,[s]
network_test_lineaire_steps6_nodes2_edges2_data2.json,1534258402.908965,FAIL,0.0153825283050537 11,[s]
```

```
network_test_memoisation_steps4_nodes4_edges4_data2.json,1534258402.9243896,OK,0.02161240577697754,[s]
network_test_tmp_steps8_nodes4_edges6_data3.json,1534258402.9460204,timeout,inf,[s]
network_test_memoisation_steps4_nodes2_edges0_data1.json,1534258407.9476116,FAIL,0.1221160888671875,[s]
network_test_lineaire_steps8_nodes4_edges6_data1.json,1534258408.0697837,timeout,inf,[s]
network_test_tmp_steps6_nodes3_edges4_data1.json,1534258413.0709414,FAIL,0.0024166107177734375,[s]
network_test_lineaire_steps8_nodes3_edges2_data3.json,1534258413.07339,timeout,inf,[s]
network_test_lineaire_steps6_nodes3_edges2_data3.json,1534258418.074231,timeout,inf,[s]
network_test_lineaire_steps8_nodes4_edges2_data3.json,1534258423.075012,timeout,inf,[s]
network_test_lineaire_steps4_nodes3_edges4_data2.json,1534258428.0759187,FAIL,0.0017485618591308594,[s]
network_test_tmp_steps8_nodes4_edges4_data0.json,1534258428.0777,OK,0.0170133113861084,[s]
network_test_memoisation_steps8_nodes4_edges4_data2.json,1534258428.0947747,timeout,inf,[s]
network_test_memoisation_steps8_nodes2_edges2_data0.json,1534258433.0970666,FAIL,0.002272367477416992,[s]
network_test_lineaire_steps4_nodes3_edges2_data3.json,1534258433.099353,OK,1.1625418663024902,[s]
network_test_tmp_steps4_nodes2_edges0_data3.json,1534258434.261948,OK,0.08692312240600586,[s]
network_test_tmp_steps2_nodes4_edges2_data2.json,1534258434.3489535,OK,0.008194446563720703,[s]
network_test_tmp_steps4_nodes4_edges6_data2.json,1534258434.3571794,OK,0.2124776840209961,[s]
network_test_memoisation_steps6_nodes3_edges4_data0.json,1534258434.5697126,FAIL,0.0034728050231933594,[s]
network_test_memoisation_steps8_nodes2_edges2_data3.json,1534258434.5732179,FAIL,0.8213341236114502,[s]
network_test_tmp_steps8_nodes4_edges4_data2.json,1534258435.3946095,timeout,inf,[s]
network_test_lineaire_steps4_nodes4_edges0_data1.json,1534258440.3960927,timeout,inf,[s]
network_test_memoisation_steps2_nodes4_edges4_data0.json,1534258445.3980286,OK,0.007910013198852539,[s]
network_test_lineaire_steps6_nodes4_edges4_data3.json,1534258445.405959,timeout,inf,[s]
network_test_lineaire_steps4_nodes4_edges0_data0.json,1534258450.4074461,OK,0.013883590698242188,[s]
network_test_lineaire_steps4_nodes4_edges4_data3.json,1534258450.421388,timeout,inf,[s]
network_test_tmp_steps6_nodes4_edges0_data3.json,1534258455.4226325,timeout,inf,[s]
network_test_lineaire_steps8_nodes3_edges0_data2.json,1534258460.4242065,timeout,inf,[s]
network_test_tmp_steps6_nodes3_edges0_data2.json,1534258465.42565,timeout,inf,[s]
network_test_tmp_steps6_nodes3_edges4_data2.json,1534258470.426694,FAIL,0.0022406578063964844,[s]
network_test_tmp_steps4_nodes4_edges6_data0.json,1534258470.428967,OK,0.00847315788269043,[s]
network_test_memoisation_steps4_nodes2_edges0_data0.json,1534258470.437453,FAIL,0.009091377258300781,[s]
network_test_tmp_steps4_nodes4_edges4_data0.json,1534258470.4466286,OK,0.012592554092407227,[s]
network_test_tmp_steps4_nodes2_edges0_data1.json,1534258470.459261,FAIL,0.01119375228881836,[s]
network_test_memoisation_steps8_nodes3_edges4_data0.json,1534258470.470493,FAIL,0.004198789596557617,[s]
network_test_tmp_steps8_nodes3_edges4_data2.json,1534258470.4747052,FAIL,0.0021812915802001953,[s]
network_test_lineaire_steps2_nodes2_edges2_data3.json,1534258470.4769208,OK,0.0015785694122314453,[s]
network_test_tmp_steps8_nodes2_edges2_data1.json,1534258470.478851,FAIL,0.016942262649536133,[s]
network_test_lineaire_steps2_nodes2_edges2_data1.json,1534258470.4958684,FAIL,0.0022385120391845703,[s]
network_test_tmp_steps8_nodes2_edges2_data3.json,1534258470.498124,FAIL,0.06963753700256348,[s]
network_test_lineaire_steps2_nodes4_edges6_data1.json,1534258470.5678172,OK,0.36690735816955566,[s]
network_test_tmp_steps2_nodes3_edges0_data0.json,1534258470.9347785,FAIL,0.011674165725708008,[s]
network_test_lineaire_steps6_nodes3_edges0_data1.json,1534258470.9464715,timeout,inf,[s]
network_test_tmp_steps6_nodes4_edges4_data3.json,1534258475.949017,timeout,inf,[s]
network_test_memoisation_steps2_nodes3_edges4_data1.json,1534258480.9507058,FAIL,0.0017211437225341797,[s]
network_test_memoisation_steps6_nodes2_edges0_data2.json,1534258480.9524484,FAIL,1.3920435905456543,[s]
network_test_tmp_steps4_nodes3_edges4_data1.json,1534258482.344531,FAIL,0.0018434524536132812,[s]
network_test_memoisation_steps6_nodes4_edges6_data2.json,1534258482.3463888,timeout,inf,[s]
network_test_lineaire_steps2_nodes3_edges4_data2.json,1534258487.3482432,FAIL,0.0015244483947753906,[s]
network_test_tmp_steps6_nodes2_edges2_data1.json,1534258487.3498056,FAIL,0.005206108093261719,[s]
network_test_memoisation_steps4_nodes4_edges6_data2.json,1534258487.3550253,OK,0.019952058792114258,[s]
network_test_memoisation_steps2_nodes3_edges4_data0.json,1534258487.3749943,FAIL,0.0020837783813476562,[s]
network_test_lineaire_steps8_nodes4_edges6_data3.json,1534258487.377097,timeout,inf,[s]
network_test_lineaire_steps6_nodes4_edges0_data0.json,1534258492.3781462,OK,0.022153854370117188,[s]
network_test_tmp_steps8_nodes3_edges2_data1.json,1534258492.4003608,timeout,inf,[s]
network_test_lineaire_steps4_nodes2_edges2_data0.json,1534258497.4015675,FAIL,0.002616405487060547,[s]
network_test_memoisation_steps2_nodes3_edges4_data2.json,1534258497.4042413,FAIL,0.002228260040283203,[s]
network_test_tmp_steps8_nodes3_edges2_data2.json,1534258497.4065084,timeout,inf,[s]
network_test_lineaire_steps8_nodes2_edges0_data2.json,1534258502.4078946,FAIL,0.1830437183380127,[s]
network_test_lineaire_steps6_nodes3_edges4_data0.json,1534258502.5910096,FAIL,0.0031623840332031255,[s]
network_test_memoisation_steps2_nodes3_edges2_data3.json,1534258502.5942037,OK,0.0035183429718017578,[s]
network_test_lineaire_steps2_nodes2_edges0_data2.json,1534258502.5977337,OK,0.0016076564788818356,[s]
network_test_lineaire_steps6_nodes3_edges2_data2.json,1534258502.5993717,timeout,inf,[s]
network_test_lineaire_steps8_nodes4_edges4_data3.json,1534258507.6005342,timeout,inf,[s]
network_test_lineaire_steps2_nodes2_edges0_data3.json,1534258512.6019297,OK,0.0016410350799560547,[s]
network_test_lineaire_steps6_nodes4_edges2_data0.json,1534258512.603603,OK,0.020330905914306644,[s]
network_test_lineaire_steps4_nodes4_edges4_data2.json,1534258512.623971,timeout,inf,[s]
network_test_tmp_steps2_nodes4_edges0_data3.json,1534258517.6252942,OK,0.007477760314941406,[s]
network_test_tmp_steps8_nodes2_edges0_data2.json,1534258517.6328034,FAIL,0.16649389266967773,[s]
network_test_memoisation_steps4_nodes2_edges2_data2.json,1534258517.7993374,FAIL,0.019325733184814453,[s]
network_test_memoisation_steps8_nodes4_edges2_data3.json,1534258517.8186855,timeout,inf,[s]
network_test_lineaire_steps4_nodes3_edges0_data0.json,1534258522.8207552,FAIL,0.026768207550048828,[s]
network_test_memoisation_steps4_nodes2_edges2_data1.json,1534258522.8475988,FAIL,0.009721040725708008,[s]
network_test_memoisation_steps6_nodes3_edges2_data1.json,1534258522.857341,timeout,inf,[s]
network_test_tmp_steps2_nodes4_edges6_data2.json,1534258527.8583744,OK,0.006714582443237305,[s]
network_test_memoisation_steps2_nodes2_edges2_data3.json,1534258527.8651228,OK,0.0022954940795898438,[s]
network_test_memoisation_steps6_nodes3_edges0_data3.json,1534258527.8674767,timeout,inf,[s]
network_test_memoisation_steps2_nodes2_edges0_data2.json,1534258532.8691065,OK,0.0028195381164550785,[s]
network_test_memoisation_steps2_nodes4_edges4_data1.json,1534258532.8719637,OK,0.007576227188110352,[s]
network_test_lineaire_steps2_nodes4_edges0_data0.json,1534258532.8795774,OK,0.009897947311401367,[s]
network_test_lineaire_steps2_nodes3_edges2_data0.json,1534258532.8895166,FAIL,0.011719226837158203,[s]
network_test_memoisation_steps6_nodes4_edges0_data0.json,1534258532.9012609,timeout,inf,[s]
network_test_memoisation_steps6_nodes3_edges0_data1.json,1534258537.9028084,timeout,inf,[s]
network_test_tmp_steps4_nodes4_edges0_data2.json,1534258542.904446,OK,0.026413679122924805,[s]
network_test_lineaire_steps6_nodes3_edges2_data1.json,1534258542.9308774,FAIL,3.240157127380371,[s]
network_test_lineaire_steps4_nodes4_edges4_data0.json,1534258546.1710908,OK,0.008947849273681644,[s]
network_test_lineaire_steps8_nodes3_edges4_data0.json,1534258546.1800697,FAIL,0.004777431488037109,[s]
network_test_memoisation_steps6_nodes3_edges2_data3.json,1534258546.184908,timeout,inf,[s]
network_test_lineaire_steps8_nodes2_edges0_data0.json,1534258551.1863213,FAIL,0.006634235382080078,[s]
network_test_lineaire_steps6_nodes4_edges6_data1.json,1534258551.192991,timeout,inf,[s]
network_test_memoisation_steps2_nodes4_edges2_data0.json,1534258556.1938736,OK,0.005958080291748047,[s]
network_test_lineaire_steps4_nodes3_edges0_data1.json,1534258556.1998436,OK,0.6382021903991699,[s]
network_test_lineaire_steps4_nodes3_edges4_data3.json,1534258556.838081,FAIL,0.0016477108001708984,[s]
network_test_memoisation_steps6_nodes4_edges2_data2.json,1534258556.8397613,timeout,inf,[s]
network_test_lineaire_steps4_nodes2_edges0_data1.json,1534258561.841364,FAIL,0.01455831527709961,[s]
network_test_tmp_steps4_nodes4_edges0_data0.json,1534258561.8559434,timeout,inf,[s]
network_test_tmp_steps2_nodes4_edges2_data0.json,1534258566.856228,OK,0.010141134262084961,[s]
network_test_lineaire_steps6_nodes4_edges4_data1.json,1534258566.866408,timeout,inf,[s]
network_test_tmp_steps8_nodes4_edges6_data1.json,1534258571.8678048,timeout,inf,[s]
network_test_memoisation_steps6_nodes2_edges0_data0.json,1534258576.869282,FAIL,0.013608217239379883,[s]
network_test_tmp_steps2_nodes3_edges0_data3.json,1534258576.8829463,OK,0.004863739013671875,[s]
network_test_lineaire_steps2_nodes4_edges6_data0.json,1534258576.88783,OK,0.008908987045288086,[s]
network_test_tmp_steps6_nodes4_edges2_data0.json,1534258576.8967624,timeout,inf,[s]
network_test_tmp_steps2_nodes2_edges0_data2.json,1534258581.8970776,OK,0.004108905792236328,[s]
network_test_tmp_steps6_nodes3_edges2_data0.json,1534258581.9012382,FAIL,0.016342639923095703,[s]
```

```
network_test_tmp_steps6_nodes4_edges2_data2.json,1534258581.9176059,timeout,inf,[s]
network_test_lineaire_steps4_nodes3_edges2_data2.json,1534258586.9187915,OK,0.3052990436553955,[s]
network_test_tmp_steps2_nodes2_edges0_data0.json,1534258587.2241476,FAIL,0.0016758441925048828,[s]
network_test_memoisation_steps6_nodes4_edges0_data3.json,1534258587.225857,timeout,inf,[s]
network_test_memoisation_steps2_nodes2_edges2_data1.json,1534258592.2276866,FAIL,0.005255937576293945,[s]
network_test_memoisation_steps6_nodes4_edges6_data0.json,1534258592.2329984,OK,0.010898828506469727,[s]
network_test_memoisation_steps6_nodes4_edges4_data0.json,1534258592.2439208,OK,0.012046813964843750,[s]
network_test_tmp_steps6_nodes4_edges6_data3.json,1534258592.2560503,timeout,inf,[s]
network_test_memoisation_steps6_nodes2_edges0_data1.json,1534258597.2573776,FAIL,0.2781238555908203,[s]
network_test_memoisation_steps6_nodes3_edges4_data3.json,1534258597.53556,FAIL,0.0018210411071777344,[s]
network_test_tmp_steps8_nodes3_edges4_data1.json,1534258597.5374131,FAIL,0.0026464462280273438,[s]
network_test_tmp_steps4_nodes2_edges2_data0.json,1534258597.5400908,FAIL,0.0012791156768798828,[s]
network_test_tmp_steps4_nodes4_edges0_data1.json,1534258597.541402,OK,0.5338530540466309,[s]
network_test_tmp_steps4_nodes4_edges2_data0.json,1534258598.0752726,timeout,inf,[s]
network_test_lineaire_steps2_nodes4_edges2_data2.json,1534258603.0762217,OK,0.008859634399414062,[s]
network_test_tmp_steps6_nodes4_edges4_data2.json,1534258603.085117,timeout,inf,[s]
network_test_lineaire_steps4_nodes4_edges2_data0.json,1534258608.0864341,OK,0.015818595886230470,[s]
network_test_tmp_steps8_nodes3_edges0_data0.json,1534258608.102299,FAIL,0.037898302078247070,[s]
network_test_lineaire_steps2_nodes3_edges2_data1.json,1534258608.1402361,OK,0.027899265289306640,[s]
network_test_memoisation_steps6_nodes2_edges2_data0.json,1534258608.1681714,FAIL,0.0017716884613037110,[s]
network_test_tmp_steps2_nodes2_edges2_data0.json,1534258608.1699748,FAIL,0.0009748935699462891,[s]
network_test_memoisation_steps2_nodes3_edges0_data2.json,1534258608.1709812,OK,0.0039889812469482420,[s]
network_test_tmp_steps2_nodes4_edges6_data3.json,1534258608.1749816,OK,0.009404182434082031,[s]
network_test_tmp_steps8_nodes4_edges2_data3.json,1534258608.1844609,timeout,inf,[s]
network_test_memoisation_steps2_nodes3_edges0_data0.json,1534258613.18604,FAIL,0.24876046180725098,[s]
network_test_lineaire_steps8_nodes3_edges2_data2.json,1534258613.4348428,timeout,inf,[s]
network_test_lineaire_steps8_nodes4_edges0_data0.json,1534258618.4356577,OK,0.064625501632690430,[s]
network_test_memoisation_steps4_nodes2_edges0_data3.json,1534258618.5003393,OK,0.035064697265625,[s]
network_test_lineaire_steps6_nodes4_edges4_data0.json,1534258618.5354311,OK,0.013251066207885742,[s]
network_test_lineaire_steps8_nodes4_edges6_data0.json,1534258618.5487375,OK,0.033965110778808594,[s]
network_test_lineaire_steps2_nodes3_edges2_data2.json,1534258618.58276,OK,0.003034830930933789,[s]
network_test_lineaire_steps4_nodes4_edges2_data3.json,1534258618.585829,timeout,inf,[s]
network_test_lineaire_steps2_nodes4_edges4_data2.json,1534258623.5890024,OK,0.0074121952056884766,[s]
network_test_memoisation_steps4_nodes3_edges0_data3.json,1534258623.5964518,timeout,inf,[s]
network_test_tmp_steps8_nodes4_edges0_data2.json,1534258628.597787,timeout,inf,[s]
network_test_tmp_steps4_nodes4_edges2_data3.json,1534258633.5980942,OK,0.1289064884185791,[s]
network_test_memoisation_steps4_nodes4_edges4_data0.json,1534258633.7270558,OK,0.010926246643066406,[s]
network_test_tmp_steps2_nodes3_edges4_data3.json,1534258633.7380376,FAIL,0.0019664764404296875,[s]
network_test_memoisation_steps8_nodes3_edges0_data0.json,1534258633.7400417,FAIL,0.65751075744628910,[s]
network_test_memoisation_steps8_nodes4_edges2_data1.json,1534258634.3976061,timeout,inf,[s]
network_test_memoisation_steps8_nodes3_edges2_data1.json,1534258639.3994582,timeout,inf,[s]
network_test_memoisation_steps8_nodes3_edges4_data1.json,1534258644.400624,FAIL,0.0027496814727783203,[s]
network_test_memoisation_steps4_nodes4_edges2_data2.json,1534258644.403406,OK,0.016785144805908203,[s]
network_test_lineaire_steps2_nodes3_edges2_data3.json,1534258644.4202137,OK,0.003626108169555664,[s]
network_test_memoisation_steps2_nodes4_edges0_data3.json,1534258644.4238794,OK,0.010286569595336914,[s]
network_test_memoisation_steps2_nodes2_edges2_data2.json,1534258644.4342136,OK,0.0019605159759521484,[s]
```